OBJECTIVES OF JAVA

Objects in java cannot contain other objects; they can only have references to other objects. Deletion of objects will be managed by Run time system. An Object can pass a message to another object by calling instance methods of other object.

- Smallest Java Package?
  Java.applet.* having 1 class and 3 interfaces.
  Applet Class and AppletContext, AppletStub, Audioclip interfaces.

Javac requires the name of the source file that should be compiled. Where as java is requires the name of the class should run not the name of the class file.

Low-level language elements are called lexical tokens. Ex: - identifiers, operators and special characters.

Java identifier is composed with a sequence of characters where each character is a letter, digit, connection punctuation (underscore), currency symbol and shouldn't starts with a digit.

In java Octal and hexadecimal numbers are specified with 0 and 0x prefix respectively.

Delete, thrown, unsigned, exit, next are not a valid keywords in java.

- char a = 'a';
  char \u0061 = 'a';
  ch\u0061r a = 'a';

All are valid DECLARATIONS IN java. Because java follows Unicode.

- Which variable declarations will remain un-initialized unless explicitly initialized?
  a) static variables
  b) instance variables
  c) local variables == true
  d) class variables == these are known as static variables.
  e) none

Static variables in a class are initialized with default values when class is loaded if they are not explicitly initialized. Instance variables in a class are initialized with default values when class is instantiated if they are not explicitly initialized. Default value for reference variables is null.

- Is Empty File is a valid java source file? == Yes

- Which of these are valid declarations for main() method?
  a) static void main(String[] args) { /* .. */ }
  b) private static void main(String[] args) { /* .. */ }
  c) protected static void main(String[] args) { /* .. */ }
  d) public static int main(String[] args) { /* .. */ }
  e) public static void main(String args) { /* .. */ }
  f) final static public void main(String[] args) { /* .. */ } == Correct (compile and run)
  g) public int main(String[] args, int argc) { /* .. */ }
  h) public void main(String[] args) { /* .. */ }
  i) private static void main(String[] args) { /* .. */ }
  j) protected static void main(String[] args) { /* .. */ }

- What is the output of the following program?

```
class Sample
{
	public static void main(String[] args)
	{
		int a,b,c;
		b = 10;
		a = b = c = 20 ;
		System.out.println( b );
	}
}
```

output == compile and run. It will print 20.

- What is the output of the following program?

```
class Sample
{
	public static void main(String[] args)
	{
		String a,b,c;
		c = new String("mouse");
		a = new String("cat");
		b = a ;
		a = new String("dog");
		c = b ;
		System.out.println( c );
	}
}
```

output == cat

- What is the output of the following program?

```
class Sample
{
	public static void main(String[] args)
	{
		int value = - -10;
		System.out.println( value );
	}
}
```

output = 10 (example of unary operatons)

- int value = x % y ;
  value will always have the same sign as x.

- What is the output of the following program?

```
class sample
{
	public static void main(String[] args)
	{
		int x = 5;
		System.out.println(x / 0 );
```

```
        }
}
```

Output == Devide by zero expection.

- What is the output of the following program?

```
class sample
{
        public static void main(String[] args)
        {
                float x = 5;
                System.out.println( x / 0 );
        }
}
```

Output == +infinity;
[if x is +ve then +ve infinity.
if x is -ve then –ve infinity.
if x is 0.0 then NaN.]

- What is the output of the following program?

```
class Sample
{
        public static void main(String[] args)
        {
                System.out.println( 2+3 );
                System.out.println( " value is :: " + 2+3 );
                System.out.println( " value is :: " + (2+3) );
        }
}
```

Output == 5 \n value is: 23 \n value is: 5

- What is the output of the following program?

```
class Sample
{
        public static void main(String[] args)
        {
                System.out.println( 1+2+"3" );
                System.out.println( 1+"2"+3 );
                System.out.println( "1"+2+3 );
        }
}
```

Output == 33 \n 123 \n 123

- What is the output of the following program?

```
class Sample
{
        public static void main(String[] args)
        {
                byte b1 = 2 ;
                byte b2 = 2 ;
                byte b3 = b1 + b2 ;
```

```
                System.out.println( b3 );
        }
}
```

Output == compilation error (casting is required)

- What is the output of the following program?

```
class Sample
{
        public static void main(String[] args)
        {
                byte b1 = 2 ;
                b1 += 1 (or) b1++;
                System.out.println( b1);
        }
}
```

Output == 3 (in place operations won't be converted in to the 32 bit)

- What is the output of the following program?

```
class MyClass
{
        public static void main(String []args)
        {
                final int i = 100;
                byte b = i;
                System.out.println(b);
        }
}
```

Output: - 100

- What is the output of the following program?

```
class MyClass
{
        public static void main(String []args)
        {
                int i = 100;
                byte b = i;
                System.out.println(b);
        }
}
```

Output: - Compile time error

Java is 32bit machine; primitive data types that are lower than 4 bytes will be converted to 32 bit (byte, short and char values will be converted int) and operations will be done. It will not happened in case of in-place operations.

- What is the output of the following program?

```
class Sample
{
        public static void main(String[] args)
        {
                int x = (int) true ;
        }
}
```

Output == compilation error saying incompatible types

- What is the output of the following program?

```
class MyClass
{
        public static void main(String[] args)
        {
                char character = 'A';
                System.out.println( character+character);
        }
}
```

Select all the valid answers ('A' is ascii is 65)

a) AA
b) 130
c) Print unicode value which is equal to 130
d) Compile time error

Answer: - b

float ff = 10.5; causes compilation error – loss of precession
float ff = 10.5f; will compile and run with out error
float ff = 10; will compile and run with out error

- Casting between primitive and reference types is not permitted.
- Boolean values can be casted to other data values but vice versa is not possible.
- Null values can be casted to other data values but vice versa is not possible.

- What is the output of the following program?

```
class Sample
{
        public static void main(String[] args)
        {
                System.out.println(1.0 - 2.0 / 3.0 == 1.0 / 3.0 );
        }
}
```

Output == false [care should be taken for equality of float values. Float values are stored in approximation of bits. Here in this case mathematically they are equal but it returns the false. Because of bits and precision]

- == And != used for Object references of compatible objects. If you used ofr incompatible types it will give compile time error.

- Boolean equals (Object o) is overridden for object value equality in some Classes in API (String, BitSet, Date, File and Wrapper classes of primitive Data type). If it is not overridden then it acts as ==.

R == S - - -> has R and S have same reference value or not?

```
String s1 = new String ("WelCome");
String s2 = new String ("WelCome");
System.out.println (s1.equals (s2));
```

Output: - true

```
Dog d1 = new Dog("Pinkky");
Dog d2 = new Dog("Pinkky");
System.out.println (d1.equals (d2));
```

Output: - false

- What will be output of following program?

```
class MyClass
{
        public static void main(String[] args)
        {
                String a[] = { "AA" , "BB", "CC", };
                System.out.println(a.length);
        }
}
```

Output: - 3

Short Circuit evaluation: (&& and ||)

booelan flag =(expression 1 && expression 2)
If expression 1 true then only it executes the expression 2 else simply it returns the value.

booelan flag =(expression 1 & expression 2)
Irrespective of value in the expression 1, expression 2 is executed; final result will be obtained from this. But the logical value will be the same for both, but some mathematical expression values can be affected due to this.

booelan flag =(expression 1 || expression 2)
If expression 1 false then only it executes the expression 2 else simply it returns the value.

booelan flag =(expression 1 | expression 2)
Irrespective of value in the expression 1, expression 2 is executed; final result will be obtained from this. But the logical value will be the same for both, but some mathematical expression values can be affected due to this.

- Java uses the 2's compliment to stores the integer values. MSB (most significant bit) indicates the sign of the number.

- &, |, ~ Can applied for numeric values then bit-wise operations (binary Operations will) takes place.

- Shift Operators in Java are << (left shift), >>(right shift / right shift with fill operator), >>>(unsigned right shift/ right shift with zero fill). Significant difference between the >> and >>> is visual when applied to for negative numbers. For positive numbers both results the same value.

- Which of the following is not a valid operator in java?
  a) %
  b) <<<
  c) >>>
  d) %=
  e) <=

Answer: - b

- % Operator can be used for integers and float numbers.

- Given variable x of type int., which are the correct answers to double the value of x?
  a) x << 1 ;
  b) x = x * 2;
  c) x *= 2 ;
  d) x += 2;
  e) x <<= 1;

Answers: - b, c, d, e [a is wrong because it is not assigned to x]

- Given these declarations, which of the following expressions are valid?

```
Byte b = 1;
Char c = 1;
Short s = 1;
Int i = 1;
```

Select valid answers
  a) s = b * 2 ;
  b) I = b << s;
  c) B <<= s;
  d) C = c + b;
  e) S += i;

Answer: - b, c, e

[
  a , d will give compilation error for precision
  c , e are correct because of in-place operation
]

- What is the output of the following program?

```
class Sample
{
        public static void main(String[] args)
        {
                int i = 0 , j = 0 ;
                boolean t = true ;
                boolean r ;
```

```
                r = ( t &  0<(i+=1) );
                r = ( t && 0<(i+=2) );
                r = ( t |  0<(j+=1) );
                r = ( t || 0<(j+=2) );

                System.out.println( i + " " + j );
        }
}
```

Select all valid answers

a) First digit 1
b) First digit 2
c) First digit 3
d) Second digit 1
e) Second digit 2
f) Second digit 3

Answer: - c, d

- What is the output of the following program?

```
class Pizza
{
        String meat = "beef";
}

class Sample
{
        public static void main(String[] args)
        {
                Pizza p1 = new Pizza();

                System.out.println( p1.meat );

                bake(p1);

                System.out.println( p1.meat );
        }
        public static void bake(Pizza p)
        {
                p.meat = "chicken";
                p = null;
        }
}
```

Output: - Beef \n chicken [Here p = null is a formal reference only].

- Call by reference would allow values in the actual parameters to be changed via formal parameters. Formal parameter for objects is just a reference to the Object.

- Parameters variables can declared as final. These variable values can't be changed during the method. Parameter Object references can also declared as final. It means the Object references can't be changed but Object values can be changed.

Ex: -

```
public static void bake(final int x, final Pizza p)
{
        x = 10; // not allowed because it is final variable
        p.meet = " chicken"; // allowed.
        P = null; // not allowed because p is final reference
}
```

- Write a method, which takes int and return equal binary string?

```
public static String convertBinary(int x)
{
        String retstring = "";
        while ( x != 0)
        {
                int lowerBit = ( x & 1 );
                String digit = (lowerBit == 1)?"1":"0";
                retstring = digit + retstring;
                x >>>= 1 ;
        }
        return retstring;
}
```

Arrays

Arrays are Objects in java. If array is declared and create as instance of that object then all the values will be initialized to default values. If array is Object references it will be initialized to null. **In Java arrays with zero elements can be created. Length of the array can be obtained by using the attribute length = <arrayname>.length**

<element type>[]<array name> Is valid declaration of an array
<element type><array name>[] Is valid declaration of an array
<array name> = new <element type>[<no of elements>] is valid creation of array.
    If <no of elements> 0 (or) more then it is it is valid else run time exception will occur.
<array name> = { <array initialize code> }
<element type>[][]<array name> valid 2 dimensional array declaration
<element type>[]<array name>[] valid 2 dimensional array declaration
<element type><array name>[][] valid 2 dimensional array declaration

Anonymous Arrays: - arrays doesn't have name, combining the definition and instantiation of array. Basic Usage is when pass the arrays to methods. Ex: - new int[] {1,2}

- Which of the array declarations and initializations are not legal?
  a) int a[][] = new int[4][4];
  b) int []a[] = new int[4][4];
  c) int [][]a = new int[4][4];
  d) int a[][] = new int[4][];
  e) int a[][] = new int[][4];
  f) int a[] = new int[2] { 1 , 2 };
  g) int a[][] = {{ 1 , 2 } , {3} } ;
  h) int a[][] = new int[][]{{ 1 , 2 } , {3} };
  i) int a[][] = new int[][]{{ 1 , 2 } , new int[2] };
  j) int a[4] = { 1 , 2 , 3, 4};

k) int a[] = new int[-4];
l) int a[] = new int[0];
m) int a[] = new int[];

Answers: - e, f,
[
e is in correct because base array size is undefined. I.e., first array size is not defined.
f is in correct because when initialization takes place no need to give the precision.
J is in correct because left hand side never gives the precision.
K is in correct at compile time and gives runtime exception due to negative array size.
M is in correct the array size is un defined so compile time error is occurs.
]

Constructor concepts

- <constructor hearder>(< param list>) { <constructor body> }
- It should have the same name as class name
- Modifiers other than accessibility are not permitted.
- It shouldn't return a value not even void
- Constructor without any parameters is known as default constructor. This is implicit constructor provided by java.
- Once the constructor is defined with parameters then implicit constructor won't be available. If user wants he should to write it.
- **Constructors can be over loaded but not override by subclasses**.
- Constructor may not initialize all the member variables.
- Subclasses without any declared constructors would fail if the super class doesn't have default constructor.

```
class MyClass
{
        long var ;
        public void MyClass(long param)  //→ 1
        {
                var = param;
        }
        public static void main(String[] args)
        {
                MyClass a , b ;
                a = new MyClass();      //→2
                b = new MyClass(5);     //→3
        }
}
```

Select the correct answers
a) Compilation error occurs at 1 since constructors should not specify the return value
b) Compilation error occurs at 2 since class does not have default constructor
c) Compilation error occurs at 3 since class does not have constructor accepting single argument int.
d) Compile correctly.

Answer: - c

Package Concepts

- Package hierarchy represents the organization of byte code of classes and interfaces but not for the source code.
- At most one package declaration can be appear in one source file.
- This must be first statement
- To use packages through import statements; import the packages in the source files that is the first statement after the package statement.
- Import does not recursively import the sub packages of it.

Abstract Classes

- Any class can be specified with abstract keyword to indicate that it cannot be instantiated.
- A class that has an abstract method must be declared as abstract.
- If method doesn't contain body they should be declared as abstract otherwise compilation error occurs.
- "static abstract" and "final abstract" are in valid combinations.

Final classes

- A final class can't be extended (behavior can't be changed).
- A class whose definition is complete that can be declared as final.
- "final abstract" combination is not valid

Access modifiers in java are

- public :- accessed any where
- protected :- accessed with in the package classes, subclasses of the class in the same package and subclasses of same class in other packages.
- default (known as package):- accessed with in the package classes, subclasses of the class in the same package
- private :- with in its own class

Other Modifiers in java are

- **Static**: - any members declared as static belongs to the class not for instance. When class is loaded then all static variables and methods are loaded in to the memory and variables are initialized. Static methods can't access the instance members and vice versa is possible. **Class can't be static. Static variables are not serializable.**
- **Final**: - a final variable is a constant where it's value can't be changed once it is initialized. It need not be initialized at declaration but it is initialized before use. A final method in a class is complete and can't be overridden by any sub classes of the class. A final class can't be inherited.
- **Abstract**: - method does not have the implementation will be declared as abstract. **It is not applicable for attributes.**
- **Synchronized**: - only one thread at a time can execute the method of an object then method should be declared as synchronized. Then its execution is mutual exclusive among all the threads. **It is not applicable for attributes and classes. It is applicable for methods and code blocks only.**
- **Native**: - These are known as foreign methods. This implementation will be defined in other than java language. Its implementation appears elsewhere, only the method prototype is

specified in the class definition. Used in JNI technology. **It is not applicable for attributes and classes. It is not applicable for only methods.**

- **Transient**: - Objects can be stored using the serialization. There are cases where a value often instance variable will not be persistent when object is stored. Such instance variables can be specified as transient. **It is not applicable for methods and classes. Transient variables are not serializable.**
- **Volatile**: - during the execution, threads might cache the values of member variables for efficiency reasons. Since threads can share a variable, it is then vital that reading and writing of the value in the copies and the master variable do not result in any inconsistencies. As this variables value could be changed unexpectedly, the volatile modifier can be used to inform the compiler that it should not attempt to perform the optimization on the variable. **It is not applicable for methods and classes.**

- Can we specify the modifiers to the local variables?

  Answer: - we can't

- All methods in a class are implicitly passed this parameter when called?

  Answer: - Wrong.

- Only static methods can access static members?

  Answer: - Wrong because static members can be accessed by instance methods also.

- Any subclass of an abstract class must implement abstract methods?

  Answer: - Wrong it is possible that subclass is also be an abstract class.

```
class MyClass
{
        long var ;
        // Here the method name is same as class name. It is valid.
        // It is not a constructor. It is an instance method only.
        public void MyClass(long param)
        {
                var = param;
        }
        public static void main(String[] args)
        {
                MyClass a, b;
                a = new MyClass();
        }
}
```

Switch Block

Switch block will work only for Integral expressions/integer operands (char, byte, short, int). **Long is not allowed in switch**. All options (cases) are optional including the default option. The type of the case labels can be assignable to the type of the switch expression.

- class sample

```
{
```

```
        public static void main(String[] args)
        {
                if(true)
                if(false)
                System.out.println("a");
                else
                System.out.println("b");
        }
}
```

What will be the output?

Answer: - “b” will be printed.

- Switch expression type is int and case labels value of type char?

  Answer: - correct

- Switch expression type is float and case labels value of type int?

  Answer: - in Valid (float is not allow in switch expression)

- Switch expression type is byte and case labels value of type float?

  Answer: - in valid (in assignable case labels)

- Switch expression type is char and case labels value of type byte?

  Answer: - in valid (in assignable case labels) explicit casting is required.

- Switch expression type is boolean and case labels value of type Boolean?

  Answer: - in Valid (Boolean is not allow in switch expression)

For Loop

For loop structure is for (<initialization>; <loop condition>; <increment/decrement>)
<initialization> of a for loop is either a comma separated list of declaration statements or expression statements. That is in one for loop use can’t declare two different types of variables.

for(int i=10 , System.out.println(“welcome”) ; I < 10 ; I++) { } == is not valid.

int I ;
for(i=10 , System.out.println(“welcome”) ; I < 10 ; I++) { } == is valid.

Return: - it is used to stop the execution of the method and return the transfer to caller methods.

Return statement type	Void method	Non void method
Return;	Optional	Not allowed
Return <exp>;	Not allowed	Mandatory

- Boolean b = true;
  B = ! b ;

System.out.println(b);

What is the output?

Output: - false ["!" is valid]

- Select all the valid for loops
  a) int j=10; for (int i = 0, j+= 90; I < j ; I++) { j--; }
  b) for (int i = 10 ; i =0 ; i--) { }
  c) for (int i= 0 , j = 100; i < j ; j--) { ; }
  d) int i , j ; for (j = 100 ; i < j ; j--) { i+= 2; }
  e) int i = 100; for ((i>0); I++) { }

Output: - c

- if(true) { break;} is expression is right?

Output: - compilation error. [Break should be used in side loops only]

- What is the output of the following program?

```
class sample
{
        public static void main(String[] args)
        {
                for ( int i = 0 ; i < 10 ; i++)
                {
                        switch( i )
                        {
                                case 0: System.out.println ( i );
                        }
                        if ( i )
                                System.out.println ( i );
                }
        }
}
```

Answer: - compilation error. [Conditional expression of if statement is not a type of Boolean]

- Break statement can be used in side the loops and with out loops where the labeled brakes occur.

- Continue statement can be used in side the loops only.

Exceptions are objects in java. All exceptions are derived from the Throwable class. Error and its subclasses defined the exceptions those never explicitly caught and usually irrecoverable (like linkage errors, JVM errors). Run time exceptions are usually program faults.

Checked Exceptions: - All exceptions other than the Runtime exceptions and errors are known as checked exceptions. Compiler ensures that if a method can throw a checked exception (directly / indirectly) method explicitly deals with it (or) send it to caller method.

Unchecked Exceptions: - Exceptions defined by the Error (or) RuntimeException class and their subclasses are known as unchecked exceptions. Method not deals with these kinds of exceptions.

All **user-defined exceptions** are the subclasses (or) extend of the Exception class. All User-defined exceptions are **checked** exceptions. All User-defined exception class can have attributes and methods.

Exception thrown during the execution of **try** block can be caught and handled in the **catch** block. The code in **finally** block always will be executed. Catch blocks (zero or more) (or) finally block must always appear with conjunction with a try block. The compiler complains if a catch block for a super class exception shadows subclass exception, as the subclass exception catch block will never be reached. Any code in the try is executed then finally block is guaranteed to be executed regardless of any catch block is executed. On Exit of finally block, if there are any pending exceptions then method execution will be aborted. A program can throw an exception using the **throw** clause. Exception propagates to the caller methods using the **throws** clause.

- Type of the Exception must be assignable to the type of the argument in the catch block.
- Thrown Exceptions must be assignable to the type of the exceptions specified in the throws clause.
- The overridden method only needs to specify a **subset** of the exceptions the original method declares it can throw.

```
import java.io.*;
class sample
{
	public static void main(String[] args)
	{
		try
		{
			// code to read a file and display on the console
		}
		catch(Exception e)
		{
			//code
		}
		catch(IOException ie)
		{
			//code
		}
	}
}
```

What is the output?

Output: - Compile time error saying IO Exception is already caught.

```
import java.io.*;
class sample
{
	public static void main(String[] args)
	{
		try
		{
```

```
                        // code to read a file and dispaly on the console

                }
                finally
                {
                        // code
                }
                catch(Exception e)
                {
                        //code
                }
        }
}
```

What is the output?

Output: - compilation error saying, “Catch without try”

- Try block should associate with

a) One (or) more catch blocks
b) Finally block
c) Either a (or) b
d) Both

Answer: - c

- What is the output of the following program?

```
import java.io.*;
class sample
{
        public static void main(String[] args)
        {
                int k = 0 ;
                try
                {
                        int i = 5 / k ;
                }
                catch(ArithmeticException e1)
                {
                        System.out.println("1");
                }
                catch(RuntimeException e2)
                {
                        System.out.println("2");
                }
                catch(Exception e3)
                {
                        System.out.println("3");
                }
                finally
                {
                        System.out.println("4");
                }
                System.out.println("5");
```

```
        }
}
```

Output: - 1 \n 4 \n 5

- What is the output of the following program?

```
class sample
{
        public static void main(String[] args)
        {
                try
                {
                        if ( args.length == 0 ) return;
                        System.out.println (args[0]);
                }


                finally
                {
                        System.out.println ("The End");
                }
        }
}
```

Select valid answers:

a) If run with no arguments, the program will produce no output
b) If run with no arguments, the program will print “The End”
c) The program will throw an IndexOutOFBoundsException
d) If run with one argument, the program will simply print given argument.
e) If run with one argument, the program will simply print given argument followed by “The End”

Output: - b, e

```
class sample
{
        public static void main(String[] args)
        {
                RuntimeException re = null;
                throw re;
        }
}
```

Select right answer

a) Fail to compile since main method does not declare that it throws RuntimeException in its declaration
b) Fail to compile, since it cannot throw Re
c) Compile with out error and will throw RuntimeException when run.
d) Compile with our error and will throw NullPointerException when run.
e) Compile with our error and will run without any output

Answer: - d

- An overriding method must declare that it throws the same exceptions classes as the method it overrides?

Answer: - Wrong. The override one can declare any subset of above method exceptions declared at method signature (or) any subclasses of declared exceptions.

```
class sample
{
	public static void main(String[] args)
	{
		try
		{
			f() ;
		}
		catch(InterruptedException e1)
		{
			System.out.println("1");
			throw new RuntimeException();
		}
		catch(RuntimeException e2)
		{
			System.out.println("2");
			return;
		}
		catch(Exception e3)
		{
			System.out.println("3");
		}
		finally
		{
			System.out.println("4");
		}
		System.out.println("5");
	}
	public static void f() throws InterruptedException
	{
		throw new InterruptedException("Time for Lunch");
	}
}
```

What will be the ouput?

Output: - 1 \n 4 \n RuntimeException

```
class sample
{
	public static void main(String[] args) throws InterruptedException
	{
		try
		{
			f() ;
			System.out.println("1");
		}
		finally
		{
			System.out.println("2");
		}
```

```
                System.out.println("3");
        }
        public static void f() throws InterruptedException
        {
                throw new InterruptedException("Time for Lunch");
        }
}
```

What is the Output?

Output: - 2 \n InterruptedException

```
class sample
{
        void f() throws InterruptedException, ArithmeticException
        {
                div(5,5);
        }
        int div(int i , int j) throws ArithmeticException
        {
                return i/j ;
        }
}
class sample1 extends sample
{
        void f() //throws ... list
        {
                try
                {
                        div(5,0);
                }
                catch(ArithmeticException e)
                {
                        return;
                }
                throw new RuntimeException("  Runtime Exception " );
        }
}
```

Select valid answers for override f() exception list

a) does not need to specify any exceptions
b) needs to specify that throws ArithmeticException
c) needs to specify that throws InterruptedException
d) needs to specify that throws RuntimeException
e) needs to specify that throws both ArithmeticException and InterruptedException

Answer: - a

```
class sample
{
        void f() throws RuntimeException
        {
        }
}
class sample1 extends sample
```

```
{
        void f() //throws ... list
        {
        }
}
```
Select all the valid answers

a) Sample1 f() need not throw any exception
b) Sample1 f() throws RuntimeException
c) Sample1 f() throws ArithmeticException
d) Sample1 f() throws InterruptedException

Answer: - a, b, c

[
a, b are correct because of subset of super class exception list of f()
c is correct because it is a subclass of one of the super class exception list
d is wrong because it is neither a subset of above list or sub class of any one of above list.]

```
class Sample
{
        public static void main(String[] args)
        {
                System.out.println ("Before Try ");
                try
                {
                }
                catch(java.io.IOException e)
                {
                        System.out.println( " inside IO Exception ");
                }
                System.out.println ("end of execution ");
        }
}
```
What is the Output?

Output: - Compile time error saying that IO Exception never thrown in the body of the try block

```
class Base
{
        Base() throws Exception
        {
                throw new Exception("Sample IO Exception");
        }
}
class Sub extends Base
{}
```
How to avoid the Base Class Exception

a) Avoid the Direct Calls to the Base class
b) Write the constructor of Sub class with try and catch blocks.
c) Write the Sub class constructor with throws Exception clause.
d) None

Answer: - c

Object Oriented Programming

- If no extends clause is specified in the header of the class definition then it implicitly inherits from the java.lang.Object class.
- In java a class can inherit only one immediate super class. Java supports liner- inheritance.
- **Inheritance** defines "**is-a**" relation ship between the super and sub classes.
- **Aggregation** defines the relationship between the instance of a class and its constituents. This relationship is noted as "**has-a**".
- **A Sub class reference can be assigned to super class reference**, because a subclass object can be used where a super class object is used. If user is tempted to invoke exclusive methods of subclasses on super class reference then compiler will give error.
- At compile time, compiler will consider the left hand side object type.
- At runtime, JRE consider the Right hand side Object type.
- **A super class reference can be assigned to sub class reference with explicit cast**. It is valid at compile time. It is not valid at runtime.
- **instanceOf** operator will be used to determine the Object type in runtime.

Over loading concepts

- Over loading methods must have same name.
- Methods with overloaded names are effectively independent methods --using the same name is really just a convenience to the programmer.
- Return type, accessibility, and exception lists may vary freely.
- Overloaded methods must have different argument lists.

Void methodA(void) == wrong declaration

Overriding concepts

- Overriding methods must have same name.
- Overriding methods must have argument lists of identical type and order.
- The return type of an overriding method must be identical to that of the method it overrides.
- The accessibility must not be more restricted than the original method.
- The method must not throw checked exceptions of classes that are not possible for the original method.
- The new method definition of the subclass can only specify **all** (or) a **subset** (or) **Subclasses** of the exceptions that are specified in the original method **throws** clause.
- An overridden method can be invoked from within the subclass using the construction super.xxx()
- If Super class method parameter has **final** modifier then in Overriding method no need to have the **final** modifier. Similarly vice versa. But data type and order should remain same.
- **Late Binding (or) polymorphism concept is only for methods not for attributes**.
- **Any private, static and final methods cannot be overridden**. Overridden is for instance a method only.

```
class Base
{
        int x = 10;
        void meth()
        {
                System.out.println ("In Base Method");
        }
```

```
}

class Sub extends Base
{
        int x = 20;
        void meth()
        {
                System.out.println ("In Sub Method");
        }
        public static void main(String[] args)
        {
                Base b = new Sub();
                System.out.println (b.x);
                b.meth();
        }
}
```

Output 10 \n In Sub Method [Late Binding concept is only for methods not for attributes.]

Object reference this, super

- "this" reference is passed as an implicit parameter when an instance method is invoked. It denotes the object on which the method is invoked.
- "super" can be used in the body of an instance method in a subclass to access variables / methods of super class.
- "this" is used in the constructors then it should be the first statement in that constructor; it is used to invoke the overloaded constructor of the same class.
- "super" is used in the constructors then it should be the first statement in that constructor; it is used to invoke the immediate super class constructor.
- Subclasses without any declared constructors would fail if the super class doesn't have default constructor.
- "this" and "super" statements cannot be combined.

- Given classes A, B, and C; where B extends A and C extends B and where all the classes implement the instance method void doIt(). How can the doIt() method in Class A called from an instance methods in Class C.
  Select valid answers
  a) doIt();
  b) super.doIt();
  c) super.super.doIt();
  d) this.super.doIt();
  e) a. this.doIt();
  f) ((A) this).doIt();
  g) It is not possible.

Answer: - g. [not possible]

- Given the following code, which of these constructors could be added to the MySub class without causing the compile time error?

```
class MySuper
{
    int number;
    MySuper( int i )
```

```
        {
                number = i ;
        }
    }
    class MySub extends MySuper
    {
        int count;
        MySub( int cnt , int num)
        {
                super(num);
                count = cnt;
        }
        // Insert additional contractors here
    }
```

Select all the valid answers

a) MySub(){}
b) MySub(int cnt){ count = cnt;}
c) MySub(int cnt){super();count = cnt; }
d) MySub(int cnt){ count = cnt; super(cnt); }
e) MySub(int cnt){this(cnt, cnt);}
f) MySub(int cnt){super(cnt); this(cnt,cnt);}

Answer: - e [f is in correct due to " this & super can't be combined]

Interface Concepts

- The methods in an interface are all abstract.
- Interfaces can't be specified as abstract.
- Default Access modifier for the interface methods are "public".
- All interface methods can't be static.
- All interface methods can't be final.
- **Interface can extend any number of interfaces.**
- All interface variables are "public static final".
- Interface variables can be used with its fully qualified names regardless of implementing (or) extending the interface.
- Interface References can be declared and assigned to implemented class.
- **Liner inheritance between the classes**: - class can extends only one other class
- **Multiple inheritances between the interfaces**: - an interface can extend any number of other interfaces.
- **Multiple inheritances between the interfaces and classes**: - a class implements any number of interfaces.

- Which of these statements about interface are true? Select all the valid answers
  a) Interface permit multiple implementation inheritance
  b) Interface can be extended by any number of interfaces
  c) Interface can extends any number of interfaces
  d) Members of an interface never be static
  e) Members of an interface never ALLWAYS be static

Answer: - b, c

- Given the following variable declaration in the interface. Which of these declarations are equal to it?

```
int answer = 42;
```

a) public static int answer = 42;
b) public final int answer = 42;
c) static final int answer = 42;
d) public int answer = 42;
e) final int answer = 42;

Answer: - all are correct

- Which of these statements concern about interfaces are correct? Select all valid answers
  a) The Keyword extends is used to signify that an interface inherits from another interface
  b) The Keyword extends is used to signify that a class inherits from another interface
  c) The Keyword implements is used to signify that an interface inherits from another interface
  d) The Keyword implements is used to signify that a class inherits from an interface
  e) The Keyword implements is used to signify that a class inherits from another class

Answer: - a, d

- What is wrong in the following code?

```
interface Interface1
{
        int VAL_A = 1 ;
        int VAL_B = 2 ;
        void f();
        void g();
}
interface Interface2
{
        int VAL_B = 3 ;
        int VAL_C = 4 ;
        void g();
        void h();
}
abstract class MyClass implements Interface1, Interface2
{
        void f() {};
        void g() {};
}
```

Select the right answer

a) Interface1 and Interface2 do not match; therefore MyClass cannot implement them both.
b) MyClass only implements Interface1. Implementation for void h() from Interafce2 is missing.
c) The declarations of void g() in the 2 interfaces clash.
d) The definition of VAL_B in the 2 interfaces clash.
e) Nothing is wrong with the code. It will compile without errors.

Answer: - e

- Reference Assignments are generally permitted “up” [super = sub] the inheritance hierarchy with implicit conversion of Source reference to destination reference. “Down” [sub = super] inheritance hierarchy with explicit conversion of Source reference to destination reference.

InstanceOf: - the instanceOf Operator effectively determines the LHS reference is an instance of RHS reference or not. It returns true that LHS reference can be cast to RHS reference. If it is false then it raises ClasscastException at Runtime. It has 2 checks. One is at Compilation time and other is at runtime. Compilation check is “source reference can assign to destination reference I.e., both are compatible reference types are not.” A Runtime Actual object takes place in the comparison process.

- Given the following program, which statement is true?

```
class A {}
class B extends A {}
public class sample
{
        public static void main(String[] args)
        {
                A[] arrA;
                B[] arrB;
                arrA = new A[10];
                arrB = new B[10];
                arrA = arrB ;                   // --> 1
                arrB = (B[]) arrA;              // --> 2
                arrA = new A[10];
                arrB = (B[]) arrA;              // --> 3
        }
}
```

a) Fail to compile at mark 1
b) Will throw ClassCastException at mark 2
c) Will throw ClassCastException at mark 3
d) Compile and run without problems even if (B[]) cast is removed from mark 2 ,3
e) Compile and run without problems but would not do so if the if (B[]) cast lines were removed from mark 2 ,3

Answer: - c

- Which is the first line that will cause compilation error in the following program?

```
class MyClass
{
        public static void main(String[] args)
        {
                MyClass a;
                MySubClass b;

                a = new MyClass();              // --> 1
                b = new MySubClass();           // --> 2
                a = b ;                         // --> 3
                b = a ;                         // --> 4
                a = new MySubClass();           // --> 5
                b = new MyClass();              // --> 6
        }
```

```
}
class MySubClass extends MyClass {}
```

a) line labeled 1
b) line labeled 2
c) line labeled 3
d) line labeled 4
e) line labeled 5
f) line labeled 6

Answer: - d

- Given the following definitions and reference declarations, which of the following assignments are legal?

```
//definitions
interface I1{}
interface I2{}
class C1 implemnts I1 {}
class C2 implemnts I2 {}
class C3 extedns C1 implemnts I2 {}

//Reference declartions
C1 obj1;
C2 obj2;
C3 obj3;
```

Select all the valid answers

a) obj2 = obj1;
b) obj3 = obj1;
c) obj3 = obj2;
d) I1 a = obj2;
e) I1 b = obj3;
f) I2 c = obj1;

Answer: - e

- Given the following class definitions and the following reference declarations, what can be said about the statement **y = (sub) x**;

```
Class Super {}
Class Sub extends Super {}

// Reference declarations
        Super x;
        Sub y;
```

Select the right answers

a) Illegal at compile time
b) Legal at compile time, might be illegal at runtime
c) Definitely legal at runtime, but cast was not strictly needed
d) Definitely legal at runtime, and cast was needed

Answer: - b

- Which letters will be printed when you run the following program?

```
class A {}
class B extends A {}
class C extends B {}
class D extends C {}
class sample
{
        public static void main(String[] args)
        {
                B b = new C();
                A a = b ;
                if ( a instanceof A )  System.out.println("A");
                if ( a instanceof B )  System.out.println("B");
                if ( a instanceof C )  System.out.println("C");
                if ( a instanceof D )  System.out.println("D");
        }
}
```

Select all the valid answers:

a) A will be printed
b) B will be printed
c) C will be printed
d) D will be printed

Answer: - a, b, c

- Given classes A, B, C, where B is a subclass of A and C is a subclass of B, which one of these Boolean expressions correctly identifies when an object o has actually been instantiated from class B as opposed to From A or C?

Select all the right answers

a) (o instanceof B) && (! (o instanceof A))
b) (o instanceof B) && (! (o instanceof C))
c) ! ((o instanceof A) || (o instanceof B))
d) (o instanceof B)
e) (o instanceof B) && ! ((o instanceof A) || (o instanceof C))

Answer: - b, d

- What will be the output of the following program?

```
interface I {}
interface J {}
class C implements I {}
class D extends C implements J {}
class sample
{
        public static void main(String[] args)
        {
                I x = new D();
                if ( x instanceof I )  System.out.println("I");
                if ( x instanceof J )  System.out.println("J");
                if ( x instanceof C )  System.out.println("C");
                if ( x instanceof D )  System.out.println("D");
        }
}
```

Select all the valid answers:

a) I will be printed
b) J will be printed
c) C will be printed
d) D will be printed

Answer: - a, b, c, d

- What will be the output of the following program?

```
class A {}
class B extends A {}
class C extends B {}
class D extends A {}
class Sample
{
        public static void main(String[] args)
        {
                A a1,a2,a3;
                if (( (a1 = new D()) instanceof C ) && ( (a2 = new C()) instanceof A ))
                        System.out.println ("1");
                if (( (a1 = new C()) instanceof C ) && ( (a2 = new C()) instanceof B ))
                        System.out.println ("2");
                if (( (a3 = new B()) instanceof C ) && ( (a2 = new C()) instanceof A ))
                        System.out.println ("3");
        }
}
```

Output is 2

- What will be the result of attempting to compile and run the following program?

```
class A
{
        int f() { return 0 ; }
}
class B extends A
{
        int f() { return 1 ; }
}
class C extends B
{
        int f() { return 2 ; }
}
public class sample
{
        public static void main(String[] args)
        {
                A ref1 = new C();
                B ref2 = (B) ref1;
                System.out.println(ref2.f());
        }
}
```

Select right answer

a) Fail to compile

b) Compile without error and throws ClassCastException when run
c) Compile without error and prints 0 when run
d) Compile without error and prints 1 when run
e) Compile without error and prints 2 when run

Answer: - e

- What will be the result of attempting to compile and run the following program?

```
class A
{
        private int f() { return 0 ; }
        public int g() { return 3; }
}
class B extends A
{
        private int f() { return 1 ; }
        public int g() { return f(); }
}
class C extends B
{
        public int f() { return 2 ; }
}
public class sample
{
        public static void main(String[] args)
        {
                A ref1 = new C();
                B ref2 = (B) ref1;
                System.out.println (ref2.g());
        }
}
```

Select right answer

a) Fail to compile
b) Compile without error and prints 0 when run
c) Compile without error and prints 1 when run
d) Compile without error and prints 2 when run
e) Compile without error and prints 3 when run

Answer: - c

- Given the following code, which statements are true?

```
interface HavenlyBody
{
        String describe();
}
class Star
{
        String starName;
        public String describe()
        {
                return "start "+ starName;
        }
}
```

```
class Planet extends Star
{
        String name;
        public String describe()
        {
                return "Planet "+ name +" orbiting star "+ starName;
        }
}
```

Select all valid answers

a) Fail to compile
b) Use of inheritance is justified since planet is-a star
c) Fail to compile if the name ‘starName’ is replaced with the name ‘bodyName’ through out the Star class definition
d) Fail to compile if the name ‘starName’ is replaced with the name ‘name’ through out the Star class definition
e) An instance of planet is a valid instance of HavenlyBody

Answer: - c, d

- Given the following code, which statements are true?

```
interface HavenlyBody
{
        String describe();
}

class Star implements  HavenlyBody
{
        String starName;
        public String describe()
        {
                return "start "+ starName;
        }
}
class Planet
{
        String name;
        Star obtaining;
        public String describe()
        {
                return "Planet "+ name +" orbiting "+ obtaining.describe();
        }
}
```

Select all valid answers

a) Fail to compile
b) Use of aggregation is justified with planet has-a star
c) Fail to compile if the name ‘starName’ is replaced with the name ‘bodyName’ through out the Star class definition
d) Fail to compile if the name ‘starName’ is replaced with the name ‘name’ through out the Star class definition
e) An instance of planet is a valid instance of HavenlyBody

Answer: - b

- How do you verify that port is bind (in use) or not?

Answer: - Create a Server Socket on that port. If it is create then it is un-bind. If it is in use then it will throw Bind Exception of IO type.

- What is the output of the following program?

```
class Sample
{
        static String s1;
        static String s2 ;
        public static void main(String[] args)
        {
                String s3 = s1 + s2;
                System.out.println (s3);
        }
}
```

Output nullnull

- What is the output of the following program?

```
class Sample
{
        public static void main(String[] args)
        {
                for ( int i = 0 , j = 10 ; i < j ; i++ , j-- )
                        System.out.println ( i + "\t" + j );
        }
}
```

Select valid answers

a) Compile time error
b) Runtime error
c) 0 = 10, 1 = 9, 2 = 8, 3 = 7, 4 = 6

Answer: - c

- SNMP = Simple Network management Protocol used to monitor the server.

- The java language is architecturally neutral.

- Instance variables maintain a separate value for the state information for each class instance.

- To create a class level constant, use the following two keywords together
  – final and static

- Call static methods with a class identifier.

- What is the output of the following program?

```
public class MyClass
{
        final int i ;
        public static void main(String[] arguments)
        {
                System.out.println(new MyClass().i);
```

```
        }
}
```

Answer: - Give compilation error.

- When you attempt to add a float, int and byte the result will be == float

- Java language has support for which following types of comments == block, line and javadoc

- The following piece of code includes a class variable and an instance method:

```
public class Counter
{
        int count;
        void incrementCount()
        {
                count++;
        }
        int getCount()
        {
                return count;
        }
}
```

Answer: - False it is not using any class variables.

- What results would print from the following code snippet:

    System.out.println ("12345 ". valueOf (54321));

Answer: - 54321 will be printed. The valueOf (xxx) will convert xxx to String.

- Which methods are correct to overload the aMethod() in the following code?

```
public class Test1
{
        public float aMethod(float a, float b){ }
        // add here
}
```

Select all the Answers:

a) public int aMethod(int a, int b) { }
b) public float aMethod(float a, float b) {}
c) public float aMethod(float a, float b, int c) throws _Exception { }
d) public float aMethod(float c, float d) {}
e) private float aMethod(int a, int b, int c) { }

Answer: - a, c, e

- Which methods are correct to override the aMethod() in the following code?

Consider these classes, defined in separate source files:

```
public class Test1
{
        public float aMethod(float a, float b) throws IOException {  }
}
public class Test2 extends Test1
{
```

```
        // add here
}
```
Select all the Answers:

a) float aMethod(float a, float b){}
b) public int aMethod(int a, int b) throws Exception{}
c) public float aMethod(float a, float b) throws _Exception{}
d) public float aMethod(float p, float q){}

Answer: - d [c is in correct because you don't knew the relation between the IOException and _Exception. If _Exception is a subclass of IOException then c also correct]

- The "is-a" relationship is implemented by inheritance, using the Java keyword "extends".
- The "has-a" relationship is implemented by providing the class with member variables.

Serialization

- If the following code block were within an appropriate try/catch block, would it compile without error?

```
FileInputStream fis = new FileInputStream("inputfile");
ObjectInputStream ois = new ObjectInputStream(fis);
Point p = ois.readObject();
```

Answer: - it will not compile. The readObject method of ObjectInputStream returns an Object. You must add a cast to the appropriate type (here a Point) for this to compile.

- Which method(s) must a Serializable class implement?

Answer: - The Serializable interface includes no methods. It only serves to tag a class as serializable.

- What exception(s) can be thrown when reading a serialized object?

Answer: - InvalidClassException, StreamCorruptedException, OptionalDataException are subclasses of IOException that may be thrown. There is no such thing as a SerializableException.

- What exception(s) can be thrown when serializing (writing) an object?

Answer: - NotSerializableException is the most commonly thrown exception, InvalidClassException and IOException are also possible.

- Given that you have an object with references to several data members, and those data members share multiple references to themselves, what must be done when you serialize the object, such that when you deserialize the object, all the original references are restored?

Answer: - Nothing much is required additionally. It is automatically done by java serialization. The serialization mechanism handles saving object trees with multiple and circular references such that restoring the tree will keep the original references intact.

Inner classes (Or) Nested Classes

There are 4 categories of classes in java.

1. **Top-level Nested (static) Classes and interfaces**
2. **Non-Static inner Classes**
3. **Local Classes**
4. **Anonymous Classes**

Type 1 is considered as top-level classes and Type 2, 3 and 4 are known as Inner / nested classes. An inner class is the same as any other class, but is declared inside some other class. Top-level classes can be instantiated independently where as inner classes instance should associate with enclosed class instance. **Non-Static inner Classes** can be defined as instance members of other class just like instance variable (or) Method. **Local Classes** can be defined as a block of code in side a method (or) local block. **Anonymous Classes** can be defined and instantiated "on the fly" in the expressions. **Local Classes** and **Anonymous Classes** can be **static or non-static** (instance of each class associate with enclosing class instance).

- The dollar-separated name is printed out the class name by using the methods getClass().getName() on an instance of the inner class.
- The dollar($) separated convention is adopted for inner class names to insure that there is no ambiguity between inner classes and package members.
- An inner class may be declared static.
- **Top-level nested interfaces are implicitly static.**
  Top-Level nested classes and interfaces can be nested to any depth, but only with in other static top-level classes and interfaces.

(P.T.O)

Entity	Declaration Context (where it can be defined)	Inner class Access Modifier(s)	Outer Instance association is required or not	Direct access to enclosing context	Defines static / instance members
Top-Level Nested (static) class	As static class member	All	No	Static members in the enclosing context	Both static and non-static
Non-Static inner class	As non static class member	All	Yes	All members (including the private) in the enclosing context	Only non-static
Local class (non-static)	In block with non-static context	None	Yes	All members in the enclosing context + local final variables	Only non-static
Local class (static)	In block with static context	None	No	Static members in the enclosing context + local final variables	Only non-static
Anonymous Class (non- static)	As a expression in non-static context	None	Yes	All members in the enclosing context + local final variables	Only non-static
Anonymous Class (static)	As a expression in static context	None	No	Static members in the enclosing context + local final variables	Only non-static
Interface	As a package member / static class member	Public only	N/A	N/A	Static variables and non-static method signatures

Top-level Nested Classes and interfaces ex: -

- Static inner class does not have any reference to an enclosing instance.
- You can create an instance of a static inner class without the need for a current instance of the enclosing class.

```
public class TopLevelClass
{
	// ...
	static class NestedToplevelClass
	{
		// ...
		interface NestedTopLevelInterface
		{
			// ...
		}

		static class NestedToplevelClass1
		{
			// ...
		}
	}
}
```

Non-static inner classes Ex: -

- A special form of new operator will be used to create the inner class instance.
  <Enclosing object reference>.new <inner class name ()>
- An implicit reference to the enclosing object is always available in every method and constructor of non-static inner class.

- Non-static inner classes can be nested; names of instance members in enclosing classes can be become shadowed. Then a special form this syntax is used to refer them. That is **<className>.this.<instance member>**
- Non-static inner classes can extend other classes and themselves be extended. Therefore inheritance and containment hierarchy must be considered while dealing with the members. Follow the 2 notations
  <className>.this.<instance member> for enclosing members
  super.<instance member> for super class members

```
class TopLevelClass
{
	// ...
	private int x = 10;
	public class NonStaticInnerClass
	{
		// ...
		int k = 100;
		void f()
		{
			k = k + x ; // Correct
			this.k = this.k + x ; // correct
			this.k = this.k + this.x ; // compile time error
			this.k = this.k + TopLevelClass.this.x ; // Correct
		}
	}
}
```

- What is the result of attempting to compile and run the following code?

```
public class MyClass
{
    public static void main(String[] args)
    {
        Outer objRef = new Outer();
        System.out.println( objRef.createInner().getSecret() );
    }
}
class Outer
{
    private int secret ;
    Outer()
    {
        secret = 123;
    }
    class Inner
    {
        int getSecret()
        {
            return secret;
        }
    }
    Inner createInner()
    {
        return new Inner();
    }
}
```

Select the right answer:

a) The code will fail to compile, since the class Inner cannot be declared within the class Outer.
b) The code will fail to compile, since the method createInner() cannot be allowed to pass objects of the inner class Inner to methods outside of the class Outer.
c) The code will fail to compile, since the secret variable is not accessible from the method getSecret().
d) The code will fail to compile, since the method getSecret() is not visible from the main() method in the class MyClass.
e) The code will compile without error and print 123 when run.

Answer: - e

- What will be the result of attempting to compile and run the following code?

```
public class MyClass
{
    public static void main(String[] args)
    {
        State st = new State();
        System.out.println(st.getValue());
        State.Memento mem = st.memento();
        st.alterValue();
        System.out.println(st.getValue());
        mem.restore();
```

```
                System.out.println(st.getValue());
        }
}
class State
{
        protected int val = 11;
        int getValue()
        {
                return val;
        }
        void alterValue()
        {
                val = (val + 7) % 31 ;
        }
        Memento memento()
        {
                return new Memento();
        }

        class Memento
        {
                int val;
                Memento()
                {
                        this.val = State.this.val;
                }
                void restore()
                {
                        ((State)this).val = this.val;
                }
        }
}
```

Select the right answer:

a) The Code will fail to compile, since the static main() tries to create a new instance of the inner class Sate.
b) The Code will fail to compile, since the class declaration of Sate.Memento is not visible from the main().
c) The Code will fail to compile, since the inner class Memento declares a variable with same name as variable in the Outer class State.
d) The Code will fail to compile, since the Memento constructor tries an invalid access through the State.this.val expression.
e) The Code will fail to compile, since the Memento method restore() tries an invalid access through the ((State).this).val expression.
f) The Code will compile without error and prints 11,18 and 11 when run.

Answer: - e

- What is the result of the following program compilation and run?

```
class MyClass
{
        public static void main(String[] args)
        {
                B.C obj = new B().new C();
```

```
        }
}
class A
{
        int val;
        A(int v )
        {
                val = v;
        }
}
class B extends A
{
        int val = 1;

        B()
        {
                super(2);
        }
        class C extends A
        {
                int val = 3;
                C()
                {
                        super(4);
                        System.out.println (B.this.val);
                        System.out.println (C.this.val);
                        System.out.println (super.val);
                }
        }
}
```

Select all valid answers:

a) Fail to compile
b) Compile without error and print 2, 3 and 4 in order when run
c) Compile without error and print 1, 4 and 2 in order when run
d) Compile without error and print 1, 3 and 4 in order when run
e) Compile without error and print 3, 2 and 1 in order when run

Answer: - d

- What will be the output of the following program?

```
class MyClass
{
        public static void main(String[] args)
        {
                Outer obj = new Outer();
                obj.f();
        }
}
class Outer
{
        int x = 1;
        static void f()
        {
```

```
		Inner i = new Inner();
	}
	class Inner
	{
		Inner()
		{
			System.out.println(" in side inner ");
		}
	}
}
```

Select the valid answers:

a) Fail to Compile
b) Compile with out error and prints “in side inner”
c) None

Answer: - a [non-static inner class can’t be accessed in side the static method.]

Local classes Ex: -

A local class is that is defined in a block. This could be a method body, a constructor, a local block, and a static initializer or instance initializer. Local classes are visible with in the block. Local classes never be static. If the context is static then it is implicitly static other wise it is non-static.

- Local class can’t have any static members.
- Local classes can’t have any accessibility.
- Clients outside the context of local class can’t create or access these classes directly. A local class can be instantiated in the block in which it is defined.
- A Method can return an instance of local class. Local class type must be assignable to the return type of the method. Often-super type of the local class can be specifies as return type.

- What will be the output of the following program?

```
interface IDrawable
{
	void draw();
}
class Shape implements IDrawable
{
	public void draw()
	{
		System.out.println("Drawing a shape");
	}
}
class Painter
{
	public Shape createCircle(final double radius)
	{
		class Circle extends Shape
		{
			public void draw()
			{
				System.out.println("Drawing a Circle with Radius::" + radius);
			}
		}
		return new Circle();
```

```
        }

        public static IDrawable createMap()
        {
                class Map implements IDrawable
                {
                        public void draw()
                        {
                                System.out.println("Drawing a MAP");
                        }
                }
                return new Map();
        }
}
public class MyClass
{
        public static void main(String[] args)
        {
                IDrawable obj[] = { new Painter().createCircle(5),
                                    Painter.createMap(),
                                    new Painter().createMap()};
                for ( int i = 0 ; i < obj.length ; i++ )
                {
                        obj[i].draw();
                }
                System.out.println("Local class Names :: ");
                System.out.println(obj[0].getClass());
                System.out.println(obj[1].getClass());
        }
}
```

Output: -

```
Drawing a Circle with Radius::5.0
Drawing a MAP
Drawing a MAP
Local class Names ::
class Painter$1$Circle
class Painter$1$Map
```

Anonymous Classes

- Combining the process of class definition and instantiation in to a single step.
- These classes do not have names.
- Instance of the class can only obtained with the definition of a class.
- Anonymous classes are typically used for creating the objects “on the fly” in the contexts such as return value of the method, or as an argument in a method call, or in initialization of variables.
- These can also used to extend the adapter classes.
- Static keyword is not used explicitly. Ex: - anonymous class return value of a static method would be static, as it was used to initialize a static member variable.
- Access rules for these classes are same as local classes.
- Non-static anynomous classes can access all the variables in the enclosed context and local final variables of local scope.

- Static anynomous classes can access only static variables in the enclosed context and local final variables of local scope.
- Anonymous classes should be small.
- An anonymous class can be a subclass of another explicit class, or it can implement a single explicit interface.
- An anonymous class cannot be both an explicit subclass and implement an interface.

```
interface IDrawable
{
	void draw();
}
class Shape implements IDrawable
{
	public void draw()
	{
		System.out.println("Drawing a shape");
	}
}
class Painter
{
	public Shape createShape()
	{
		return new Shape()   // == Extending the Shape class
		{
			public void draw()
			{
				System.out.println("Drawing a New Shpae");
			}
		};
	}

	public static IDrawable createDrawable()
	{
		return new IDrawable()    // == Implementing The IDrawable Interface
		{
			public void draw()
			{
				System.out.println("Drawing a IDrawable");
			}
		};
	}
}
public class MyClass
{
	public static void main(String[] args)
	{
		IDrawable obj[] = { new Painter().createShape(),
							Painter.createDrawable(),
							new Painter().createDrawable()};
		for ( int i = 0 ; i < obj.length ; i++ )
		{
			obj[i].draw();
		}
```

```
                System.out.println("Local class Names :: ");
                System.out.println(obj[0].getClass());
                System.out.println(obj[1].getClass());
        }
}
```

Output: -

```
Drawing a New Shpae
Drawing a IDrawable
Drawing a IDrawable
Local class Names ::
class Painter$1
class Painter$2
```

Given Declaration

```
interface IntHolder
{
    int getInt();
}
```

Which of the following declarations are valid?

```
// === 1 ===
        IntHolder makeIntHolder(int I)
        {
                return new IntHolder()
                {
                        public int getInt
                        {
                                return I;
                        }
                }
        }
// === 2 ===
        IntHolder makeIntHolder(final int I)
        {
                return new IntHolder
                {
                        public int getInt
                        {
                                return I;
                        }
                }
        }
// === 3 ===
        IntHolder makeIntHolder(int I)
        {
                class MyIH implements IntHolder
                {
                        public int getInt
                        {
                                return I;
                        }
                }
```

```
                return new MyIH();
        }
// === 4 ===
        IntHolder makeIntHolder(final int I)
        {
                class MyIH implements IntHolder
                {
                        public int getInt
                        {
                                return I;
                        }
                }
                return new MyIH();
        }
// === 5 ===
        IntHolder makeIntHolder(int I)
        {
                return new MyIH(i);
        }
        static class MyHI implements IntHolder
        {
                final int j;
                MyIH(int i)
                {
                        j = i ;
                }
                public int getInt()
                {
                        return j;
                }
        }
```

Select valid answers: -

a) Method labeled 1
b) Method labeled 2
c) Method labeled 3
d) Method labeled 4
e) Method labeled 5

Answer: - d, e [2 is wrong because constructor declaration is wrong.]

- Which of these statements are true for nested classes?
  a) An instance of top-level nested class has an inherit outer instance
  b) A top-level nested class can contain non-static member variables
  c) A top-level nested interface can contain non-static member variables
  d) A top-level nested interface inherit outer instance
  e) For each instance of the outer class there can exist many instances of a non-static inner class.

Answer: - b, e

Select all valid answers:

a) non-static inner classes must have either default or public accessibility.
b) All nested classes can contain other top-level nested classes.

c) Methods in all nested classes can be declared static.
d) All nested classes can be declared as static.
e) Top-Level nested classes can contain non-static methods.

Answer: - e

Which of these statements are true?

a) You cannot declare static members within a non-static inner class.
b) If a non-static inner class is nested with in a class named outer, then methods with in the non-static inner class must use the prefix Outer.this to access the members of the Outer class.
c) All member variables in any nested class must be declared final.
d) Anonymous class cannot have constructors.
e) If ObjRef were an instance of any nested class with in the class Outer, then (objref instanceof Outer) would yield true.

Answer: - a, d [e gives incompatible type compilation error.]

Which of the following statements are true?

a) Package member classes can be declared static.
b) Classes declared as members of Top-Level classes can be declared static.
c) Local classes can be declared static.
d) Anonymous classes can be declared as static.
e) No classes can be declared as static.

Answer: - b

Object Life Time – Garbage Collection

Objects Occupy memory. Garbage Collector (GC) is a mechanism for reclaiming the memory from the objects that are no longer in use and making it available for new Objects. Java provides automatic GC meaning that the JRE will take care about the memory management. Storage allocation through **new** will be administrated by the automatic GC.

- Reference of object is valid while the object in use.
- GC will not delete an Object leaving the reference. (Known as Dangling.)

An Object can tie up other resources (files, net connections), which should be freed explicitly. Object finalization provides a last option/Choice to the object for undertaken any action before its storage is reclaimed. Automatic GC class finalize () in an object, which is eligible for GC before, actually destroy the Object.

Protected void finalize() throws Throwable

It can be overridden by subclasses to take appropriate action before destroying the Object. This also can catch and throw exceptions like any other methods. However any exception is thrown but not caught by finalize method when invoked by GC is ignored. This only called once on an object regardless of being interrupted by exception during its execution.

```
Class wellbehavedClass
{
        ObjectOutputStream os;
        // ...
        protected void finalize()throws Trowable
        {
```

```
                try
                {
                        if ( os != null ) os.close();
                }
                finally
                {
                        super.finalize();
                }
        }
}
```

Finalize() may make object accessible again I.e., “resurrect” it thus avoiding the it being garbage collected. One simple technique can be used for this: “ assign the this reference to a static variable , from which it can be retrieved later”. But it is not recommended.

Certain aspects regarding automatic GC should be noted:

- There are no guarantees that the objects that are no longer in use will be garbage collected and their finalizers executed at all.
- There are no guarantees on the order in which the objects will be garbage collected (or) order of the finalizers will be executed.

EX: -

```
class SuperBlob
{
        static int idCounter;
        static int population;

        protected int bolbId ;

        public SuperBlob()
        {
                bolbId = idCounter++;
                population++;
        }
        protected void finalize() throws Throwable
        {
                super.finalize();
                --population;
        }
}
class Blob extends SuperBlob
{
        int[] fat;
        public Blob(int bloatedness)
        {
                fat = new int[bloatedness];
                System.out.println( bolbId + " : Hello : " );
        }
        protected void finalize() throws Throwable
        {
                System.out.println( bolbId + " : Bye : ");
                super.finalize();
        }
```

```
}
public class MyClass
{
        public static void main(String[] args)
        {
                int blobsRequired , blobsize ;
                blobsRequired = 5;
                blobsize= 50000;

                for ( int i = 0 ; i < blobsRequired ; i++ )
                {
                        new Blob(blobsize);
                }

                System.out.println(SuperBlob.population + " blobs alive" );
        }
}
```

Output: - [it is differ from system to system, platform dependent output.]

```
0 : Hello :
0 : Bye :
1 : Hello :
2 : Hello :
2 : Bye :
3 : Hello :
4 : Hello :
3 blobs alive
```

Java provides facility to invoke the GC explicitly. The System.gc() can be used to force GC. The System.runFinalization() can be used to run finalizers for objects eligible for GC.

Initializers

- Instance variables are initialized to initializer (or) default values when the objects are created with new Operator.
- Static variables are initialized initializer (or) default values when the class is loaded in to the memory.
- Static members can't refer the instance members and vice versa is possible.
- Instance Initilaizer expressions are executed in order which the instance member variables are defined. Similar in case of static initializer expressions.
- Logical error occurs when order initializer expressions are in correct.
- Initializer exceptions are unchecked exceptions. If any checked exception is thrown it should be handled with in the initializer only.

Ex:-

```
Class Sample
{
    int x = 10; // instance initializer
    static int y = 20 ; // static initializer
}
```

Static Blocks

- Static block will be executed only once when the class is initialized / loaded.

- Static block is not contained in any method
- A class can have any no. Of static blocks.
- Static initialization expressions and static blocks will execute in order, which they are declared.
- Static block can't have reference to a variable, which is defined after the static block definition.
- Typical use is: loaded the external libraries (or) execute the native methods.
- There is no difference in exception handling. It same as other methods.
- Static blocks can't be called explicitly.
- Static blocks can't pass exceptions to other blocks.

```
static
{
    // ...
}
```

Instance Blocks

- Java provides ability to initialize instance variables during the object creation using the instance blocks. Serve the same purpose as constructors.
- Syntax is same as local block.

```
{
    // ...
}
```

- Code of the instance block will be executed whenever the instance is created.
- Instance block can't have reference to a variable, which is defined after the static block definition.
- A class can have any no. Of instance blocks and these are executed in order they are specified in the class.
- Typical use of it is: factor out the code, which is common to all constructors of a class.
- One more typical use of it is: Anonymous classes, which cannot have constructors and therefore these instance blocks, can be used to initialize the instance variables.
- Exception handling is similar to all other methods.
- If any instance block throw an exception then all the class constructors should handled that exception otherwise exception will be thrown at runtime while creating the instance of a class.

Object state initialization**(constructor invocation process)

Object initialization involves the constructing the initial state of an object when it is created using the new Operator. The flow is as follows:

1. Instance variables are initialized to default values
2. Initialization of static member variables by executing their static initializer expressions and static blocks in the order, which they defined in the class definition.
3. Invocation of super class constructor implicitly / explicitly.
4. Initialization of instance member variables by executing their instance initializer expressions and instance blocks in the order, which they defined in the class definition.
5. Actual constructor will be invoked.

Class initialization / Interface initialization (Loading)

Class initialization should take place first before creating the any instance (or) before calling any static methods. Interface initialization only involves the execution of the any static initializer expressions for static variables, which are defined in the interface.

- Which are the statements are true?
  Select all valid answers
  a) Objects can explicitly be destroyed using the delete keyword
  b) An object will be garbage collected immediately after the last reference to the Object is removed.
  c) If Object obj1 is accessible from Object obj2 and Object obj2 is accessible from obj1, then obj1 and obj2 are not eligible for garbage Collection
  d) Once an object has become eligible for the garbage collection, it will remain eligible until it is destroyed.
  e) If an object obj1 can access an Object obj2 that is eligible for garbage collection, then obj1 is also eligible for garbage collection.

Answer: - e

- Which of these statements are true?
  Select all the valid Answers
  a) if an exception is thrown during execution of the finalize method of an object then the exception is ignored and the object is destroyed.
  b) All the Objects have finalize method.
  c) Objects can explicitly destroyed by explicitly calling the finalize method.
  d) The finalize method can be declared with any access modifier
  e) The Compiler will fail to compile code that defines an overriding finalize method that does not explicitly call the overridden finalize inherited from the super class.

Answer: - b

- Which of these statements are true?
  Select all the valid Answers
  a) The Compiler will fail to compile code that explicitly tries to call finalize method.
  b) The finalize method must be declared with protected accessibility.
  c) An Overriding finalize method in any class can always throw-checked exceptions.
  d) The compiler will allow code that overloads the finalize method name.
  e) The body of the finalize method can only access other objects that are eligible for garbage collection.

Answer: - d

- Which Statement describes guaranteed behavior of the garbage Collector and finalization mechanism?
  Select all the valid Answers
  a) Objects will not be destroyed until have no references to them.
  b) The finalize method will never be called more than once in an object.
  c) An object eligible for garbage collection will eventually be destroyed by garbage collection.
  d) If Object A became eligible for garbage collection before Object B, then Object A will be destroyed before Object B.

e) An Object Once eligible for garbage collection can never become accessible from an active part of the program.

Answer: - b

- Identify the position in the following program where the object, initially referenced with arg1 is eligible for garbage collection.

```
public class MyClass
{
        public static void main(String args[])
        {
        String msg;
        String pre = "This progarm was called with ";
        String post = " as first argument. " ;
        String arg1 = new String( (args.length>0) ? """+ args[0] + """ : "<no argument>");

        msg = arg1;
        arg1 = null;                            //(1)
        msg = pre + msg + post ;                //(2)
        pre = null;                             //(3)

        System.out.println( msg );

        msg = null;                             //(4)
        post = null;                            //(5)
        args = null;                            //(6)
        }
}
```

Select one of the Right answer.

a) After line labeled (1)
b) After line labeled (2)
c) After line labeled (3)
d) After line labeled (4)
e) After line labeled (5)
f) After line labeled (6)

Answer: - b [Before (1) arg1 and msg is referring the same object. After (1) string only msg referring the Object. At (2) msg is referenced new string with concatenation of other strings. So after (2) arg1 is eligible for GC.]

- Given the following class, which of these static initializers are legal in the indicated context?

```
Public class MyClass
{
        private static int count = 5;
        static final int STEP = 10;
        boolean alive;

        // INSERT STATIC INITIALIZER HERE
}
```

Select all valid answers.

a) static { alive = true; count = 0 ;}
b) static { STEP = count; }
c) static { count += STEP; }

d) static { ; }
e) static ;
f) static { count =1 ; }

Answer: - c, d, f [a is wrong because static block cannot access instance member. B is wrong final variable cannot changed. E is wrong because static blocks cannot ends with ";"]

- What will be the result of attempting to compile and run the following program?

```
public class MyClass
{
        public static void main(String args[])
        {
                MyClass obj = new MyClass(l);
        }

        static int i = 5;
        static int l ;
        int j = 7;
        int k ;

        public MyClass(int m)
        {
                System.out.println( i + ", "+ j + ", "+ k + ", " + l + ", "+ m );
        }

        { j = 70; l = 20 ; }
        static { i = 50 ; }
}
```

Select one right answer:

a) Code will fail to compile since the instance initializer tries to assign a value to a static member.
b) The code will fail to compile since the member variable k will be un-initialized when it is used.
c) The code will compile without error and will print 50, 70, 0, 20, 0 when run
d) The code will compile without error and will print 50, 50, 0, 20, 20 when run
e) The code will compile without error and will print 5, 70, 0, 20, 0 when run
f) The code will compile without error and will print 5, 7, 0, 20, 0 when run

Answer: - c

- Given the following class, which of these instance initializers could be inserted at the indicated location and allow the class to compile without errors?

```
Public class MyClass
{
        static int gap = 10;
        double length;
        final boolean active;

        // INSEWRT AN INSATNCE INITIALIZER HERE
}
```

select all the valid answers.

a) instance { active = true; }
b) MyClass { gap += 5 ; }
c) { gap = 5; length = (active?100:200) + gap ; }
d) {;}
e) {length = 4.2; }
f) {active = (gap > 5) ; length = 5.5 + gap ; }

Answer: - f [b, d, e is wrong because final variable is not initialized so instance can't be created.
C is wrong because active is not initialized.]

- What will be the result of attempting to compile and run the following class?

```
public class MyClass
{
        private static String msg(String msg)
        {
                System.out.println( msg );
                return msg;
        }

        public MyClass()
        {
                m = msg("1");
        }

        { m = msg("2"); }

        String m = msg("3");

        public static void main(String[] args)
        {
                Object obj = new MyClass();
        }
}
```

Select one right answer
a) The program will fail to compile
b) The program will compile without error and will print 1, 2 and 3 in that order when run
c) The program will compile without error and will print 2, 3 and 1 in that order when run
d) The program will compile without error and will print 3, 1 and 2 in that order when run
e) The program will compile without error and will print 1, 3 and 2 in that order when run

Answer: - a [compilation fail because of forward reference. Instance block makes the forward reference to the variable, which is declared after the instance block.]

- What will be the result of attempting to compile and run the following class?

```
public class MyClass
{
        private static String msg(String msg)
        {
                System.out.println( msg );
                return msg;
        }

        public MyClass()
```

```
        {
                m = msg("1");
        }

        String m = msg("3");

        { m = msg("2"); }

        public static void main(String[] args)
        {
                Object obj = new MyClass();
        }
}
```

Select one right answer

a) The program will fail to compile
b) The program will compile without error and will print 1, 2 and 3 in that order when run
c) The program will compile without error and will print 2, 3 and 1 in that order when run
d) The program will compile without error and will print 3, 2 and 1 in that order when run
e) The program will compile without error and will print 1, 3 and 2 in that order when run

Answer: - d

- What will be the result of attempting to compile and run the following class?

```
public class MyClass
{
        private static String msg(String msg)
        {
                System.out.println( msg );
                return msg;
        }

        static String m = msg("1");

        { m = msg("2"); }

        static { m = msg("3"); }

        public static void main(String[] args)
        {
                Object obj = new MyClass();
        }
}
```

Select one right answer

a) The program will fail to compile
b) The program will compile without error and will print 1, 2 and 3 in that order when run
c) The program will compile without error and will print 2, 3 and 1 in that order when run
d) The program will compile without error and will print 3, 2 and 1 in that order when run
e) The program will compile without error and will print 1, 3 and 2 in that order when run

Answer: - e

Threads

Multitasking allows several activities to occur concurrently on the computer.

- Process based multitasking: - Example running the spreadsheet and word processor.
- Thread based multitasking: - Example word processor printing and formatting the text at a time. Two independent paths of execution at run time to perform two tasks.

Java Supports thread-based multitasking.

Process Based multitasking	Thread based multitasking
Two processes cannot share the same address space.	Two Threads cannot share the same address space.
Context switching between the processes is expensive.	Context switching between the threads is inexpensive.
Communication between the two processes is expensive.	Communication between the two threads is inexpensive.

- A Thread is a path of execution within a program that is executed separately.
- Threads in the application at runtime use the common memory space so they can share data and code.
- Threads are lightweight.
- Threads can also share the process running the program.
- Threads makes the runtime environment asynchronous, allow different tasks to be performed concurrently.
- Threads in java are 2 types. One is User Threads and Daemon Threads.
- Daemon Threads exists only to serve the User threads. It is stopped if there is no more user threads are running, thus terminating the program.
- For **Standalone applications** User thread **main thread** will automatically created to execute main().
- User threads are again 2 types one id main Thread and other are child threads.
- Thread status can be changed using the setDaemon(Boolean). It must be done before the starting of the thread else it throws IllegalThreadStateException.
- For **GUI applications**, **AWT thread** will be created automatically to monitor the user interaction.
- Thread scheduling is not predictable.

Threads Creation:

Implementing the java.lang.Runnable interface
Extending the java.lang.Thread Class

```
Public interface Runnable
{
        public void run();
}
```

Code in the run() defines as independent path of execution and thereby entry and exit for the threads.

Procedure to create the Threads using the Runnable interface

1. Class implementing Runnable interface and provide the run method which will be executed by threads.

2. An object of thread will be created with an argument of Runnable implementation class reference. Ex: - r is the reference of a class, which implement the runnable interface. Thread t = new Thread(r);
3. start() is invoked on the thread object.

Procedure to create the Threads extending the Thread class

1. Override the run() from the Thread class to define Code to executed by the thread.
2. Subclass may call Thread (super class) constructor explicitly in the constructor to initialize the Thread.
3. start() is invoked on the thread object.

Synchronization: - Threads share the same common memory space. So, threads can share the Data and Code. There are critical situations where it is desirable that only one thread at a time has access to a shared resource. Java provides high-level concept synchronization to control use of shared resources.

A monitor is used to synchronize the access to shared resources. A region of code representing a shared resource can be associated with a monitor. Monitor implements the mutual exclusive locking mechanism. Monitor has following rules

- No other thread can enter in to the monitor if a thread has already acquired the monitor. Threads waiting for monitor will wait until it become available.
- When thread exits a monitor, a waiting thread is given the monitor and can process the shared resources associated with the monitor.

There are 2 ways which code can be synchronized
- Synchronized methods: - method can be executed by only one thread at a time.
- Synchronized blocks: - Code block can be executed by only one thread at a time.

Synchronized methods can also be static.

- While a thread is inside a synchronized method of an object, all other threads that are wish to execute this synchronized method or any other synchronized method of object wish to execute this synchronized method will have to wait.
- Non-synchronized methods of the object can of course be called any time by any thread.
- Synchronization of static methods in a class is independent from the synchronization of instance methods.
- A sub class decides whether the new definition of an inherited synchronized method will remain synchronized in the subclass or not.

The general form of synchronized block is

Synchronized (<object reference>) {<code block>}

The code block is usually related to the object on which the synchronization is being done. **<Object reference>** is mandatory in the Synchronized blocks. {} are also mandatory in the Synchronized blocks even if the code is single line.

Thread States
- **Running State**: - CPU is currently executing the Thread
- **Non – Runnable States**: - A thread remains in the non-runnable state until special transition moves it in to ready-to-run state.

- **Waiting State**: - when wait () is called on the running thread instance then it will comes under this state. In must be notified by another thread in order to move in to ready-to-run state.
- **Sleeping State**: - static method sleep () in thread class causes the current running thread to sleep state. It wakes up after a specified amount of time and move to ready-to-run state.
- **Blocked State**: - A running thread on executing a blocking operation requiring a resource (I / O call) will move the thread to blocking state. A thread also blocks when it fails to acquire the monitor of the object. The blocking operation must complete, before it moves in to the ready-to-run state.

- **Ready-to-Run State**: - it means that thread is eligible to Run and waiting for CPU. Thread scheduler will decides which thread gets to run when CPU is free. Static method yield () will cause the current running thread to move in to the ready-to-run state and thus relinquish the CPU for other threads.
- **Dead State**: - Thread can be dead because it has completed or it has been terminated. Once the thread in this state, which cannot be resurrected (re constructed). A new thread execution will explicitly started by calling the start ().

Thread Priorities: - Threads are assigned priorities that the thread scheduler can use to determine how the threads will be treated. The thread scheduler can use thread priorities to determine which thread gets to run. The thread scheduler usually decides to let the thread with the highest priority in Ready-to-Run state gets the CPU. This is not necessarily the thread that has been the longest time in the ready-to-run state.

Priorities are integer values rated from 1(Thread.MIN_PRIORITY) to 10(Thread.MAX_PRIORITY). Default priority is 5(Thread.NORM_PRIORITY). Thread inherits the priority from its parent thread. Priority can be set by setPriority() and get by getPriority() methods of Thread class.

Thread schedulers: -

- **Preemptive Scheduling: -** if a thread with higher priority than the current running priority thread comes to in the ready-to-run state then the current running thread will be preempted to ready-to-run state and gives the CPU to higher priority thread.
- **Round-robin scheduling: -** A running thread allowed to execute for a fixed amount of time after it moves to ready-to-run state to waits its turn to run again.

Schedulers are differing from platform to platform so, thread scheduling is un predictable.

Public static void sleep(long millis) throws InterruptedException

Wait and notify provide the means of communication between the threads that are in the synchronize on the same object. Wait(), notify() and notifyAll() must be executed on the synchronized code otherwise it results the illegalMonitorStateException.

Void wait(long timeout) throws InterruptedException
Void wait(long timeout, int nanos) throws InterruptedException
Void wait() throws InterruptedException
Void notify()
Void notifyAll()

All the above methods are defined in Object class.

Notify: - Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.
NotifyAll: - Wakes up all threads that are waiting on this object's monitor.

Boolean isAlive() - - determine whether the thread is alive or dead.
Void join() throws InterruptedException - - a call to this method will wait and not return until the thread has completed. A parent thread can use this method to wait for its child threads to complete before continuing.

- Which of the following is the correct way to start a new thread?

Select the one right answer

a) just create a new thread. Thread will start automatically.
b) Create a new Thread and call the begin() on the Thread.
c) Create a new Thread and call the start() on the Thread.
d) Create a new Thread and call the run() on the Thread.
e) Create a new Thread and call the resume() on the Thread.

Answer: - c

- When extending the Thread class to provide a thread's behavior which methods should be overridden?

Select all valid answers

a) begin()
b) start()
c) run()
d) resume()
e) behavior()

Answer: - c

- Which of the following statements are true?

Select all valid answers

a) The Class Thread is abstract
b) The Class Thread is implements Runnable interface.
c) Classes implementing the Runnable interface must define a method called start()
d) Calling the method run() on an object implementing Runnable will produce a new Thread
e) Programs terminate when the last non-daemon thread ends.

Answer: - b, e

- What will be the result of attempting to compile and run the following program?

```
public class MyClass extends Thread
{
        public MyClass(String s)
        {
                msg = s;
        }
        String msg;
        public void run()
        {
                System.out.println( msg );
```

```
        }
        public static void main(String[] args)
        {
                new MyClass("Hello");
                new MyClass("World");
        }
}
```

Select the one right answer

a) The program will fail to compile.
b) The Program will compile and will print Hello and world in that order every time the program is run.
c) The Program will compile and will print never-ending stream of Hello and world.
d) The Program will compile correctly and will print Hello and world when run, but the order is unpredictable.
e) The Program will compile Correctly and will simply terminate without any output when run.

Answer: - e

- Which of the following statements are true?

Select all valid answers:

a) No two threads can ever simultaneously execute synchronized methods on the same object.
b) Methods declared synchronized should not be recursive, since the objects monitor will not allow simultaneous invocations of the method.
c) Synchronized methods can only call other synchronized methods directly.
d) Inside synchronized method, one can assume that no other threads are currently executing a method in the same class.

Answer: - a

- Given the following program, which one of these statements is true?

```
public class MyClass extends Thread
{
        static Object lock1 = new Object();
        static Object lock2 = new Object();

        static volatile int i1, i2, j1, j2, k1, k2 ;

        public void run()
        {
                while (true)
                {
                        doit();
                        check();
                }
        }
        void doit()
        {
                synchronized(lock1) { i1++; }
                j1++;
                synchronized(lock2) { k1++; k2++; }
                j2++;
                synchronized(lock1) { i2++; }
```

```
        }
        void check()
        {
                if ( i1 != i2 ) System.out.println("i");
                if ( j1 != j2 ) System.out.println("j");
                if ( k1 != k2 ) System.out.println("k");
        }
        public static void main(String[] args)
        {
                new MyClass().start();
                new MyClass().start();
        }
}
```

Select the One Right answer:

a) The Program will fail to compile
b) One cannot be certain whether any of the letters I, j, and k will be printed during the execution
c) One can be certain that none of the letters I, j, k will ever be printed during the execution
d) One can be certain that the letters I and k will never be printed during the execution
e) One can be certain that the letter k will never printed during the execution.

Answer: - b [for each invocation of check () values are equal. only way of printing is check will execute between the first and second synchronized blocks. Even then output is unpredicted.]

- Which of these events will cause a thread to die?

Select all valid answers

a) The methods sleep() is called
b) The methods wait() is called
c) Execution of start() method ends
d) Execution of run() method ends
e) Execution of thread constructor ends.

Answer: - d

- What can be guaranteed by calling the method yield ()?

Select one right answer

a) The lower priority threads will granted CPU time.
b) The current thread will sleep for some time while some other threads are doing some work.
c) The current thread will not continue until other threads are finished with their work
d) The thread will sleep until it is notified
e) None of the Above.

Answer: - e [The exact behavior of the scheduler is not defined.]

- Where is the notify() defined?

Select one right Answer:

a) Thread
b) Object
c) Applet
d) Runnable

Answer: - b

- What will calling the notify() on an object implementing Runnable achieve?

Select the one right answer:

a) Will cause the thread executing the run() of the object to continue.
b) Will cause a thread that called the wait() while owning the monitor of the object to be enabled for running.
c) Will cause all the threads waiting for the monitor of the object to be enabled for running
d) Will cause am illegalMonitorStateException to be thrown
e) None of the above

Answer: - b

- How can you set a priority of the Thread?

Select one right answer:

a) Using the setPriority() in the Thread class
b) Give the priority as a parameter to thread Constructor.
c) Both of above
d) None of the Above

Answer: - a

- What will be the result of writing a method that attempts to call wait () without ensuring that the current thread owns the monitor of the object?

Select one right answer:

a) The code will fail to compile
b) Nothing special happen
c) An illegalMonitorStateException will be thrown whenever the method is called
d) An illegalMonitorStateException will be thrown if the method is called at a time when the current thread does not have the monitor object.
e) The thread will be blocked until it gains the monitor of the Object.

Answer: - d

- Which of these are plausible reasons why a thread might be alive, but still not be running?

Select valid answers:

a) Thread is waiting for some condition as a result of call to wait ()
b) Thread is waiting on a monitor for an object so that it may access a certain member variable of that object.
c) Thread is not the highest priority thread and is currently not granted CPU time.
d) Thread is sleeping as a result of a call to the sleep () method.

Answer: - a, c, d [b is wrong because synchronize is only for code not for member variables.]

Fundamental Classes

Java.lang.* will automatically imported in the every source file at compile time.

Lang package contains Object class which is the mother of all other classes, wrapper classes, Security classes, Class Loading classes, Thread classes, Standard Streams (system. in and System.out), String handling and mathematical functions.

- All classes directly or indirectly extend the Object class.
- Object is the root for every inheritance hierarchy.

Maker interfaces in java are as follows
- Serializable interface
- Cloneable inerface

Object Class methods are as follows:

int hashCode() = = when storing the objects in to the hash table this can be used to get unique hash value for an object.

Class getClass() == returns the runtime class object.

Boolean equals (Object o) == compare the Object references are same or not.

Protected Object clone () throws cloneNotSupportedException == to clone an object
Shallow copying: - copy all primitive and reference values are copied.
Deep copying: - clone an aggregate object by recursively cloning the constitute objects.

String toString() == textual representation of an Obejct.
<name of the class>@<hash code of object>

protected void finalize() throws Throwable == this method called before the Object is garbage collected.

Void notify ()
Void notifyAll ()
Void wait () throws InterruptException
Void wait (long timeout) throws InterruptException
Void wait (long timeout, int nanos) throws InterruptException

All are thread related methods.

Common Wrapper class Utility methods

- Each wrapper class defines static method **valueOf(String)** returns wrapper object corresponding to primitive value represented by String parameter.

- Each wrapper class defines a method **xxxValue()** which returns the primitive value XXX of wrapper object. Here xxx denotes the data Type.

- Each wrapper class overrides method **toString()** from Object class and returns a string representing the primitive value in the wrapper object.

- Each wrapper class overrides method **equals ()** from Object class and returns a Boolean. It compares the data equality.

- Each wrapper class overrides method **hashCode()** from Object class and returns a hash value based on the primitive value in the wrapper Object.

- Each Numeric wrapper class defines a static method **parseXXX(String)** which returns a primitive numeric value represented by String. This method throws numberFormatException if String is invalid.

Ex: -

```
Interger obj = Interger.valueOf("123"); == obj has the value 123.
Boolean bb = Boolean.valueOf("TRUE"); == bb has the value true.

int objValue = obj.intValue();     == 123
boolean  flag = bb.booleanValue(); == true

String objStr = obj.toString(); == "123"
String bbStr = bb.toString(); == "true"

Character ch1 = new Character('A');
Character ch2 = new Character('A');
boolean flag = ch1.equals(ch2); == true
boolean flag1 = ( ch1 == ch2 ) ; == flase

int x = Integer.parseInt("123"); ==  x is 123
```

Constants available are as follows:

Boolean.TRUE
Boolean.FALSE
Character.MIN_VALUE
Character.MAX_VALUE
<Wrapper class name>.MIN_VALUE
<Wrapper class name>.MAX_VALUE here Wrapper is only Numeric type only.

Character methods

static boolean isLowerCase(char)
static boolean isUpperCase(char)
static boolean isTitleCase(char)
static boolean isDigit(char)
static boolean isLetter(char)
static boolean isLetterOrDigit(char)
static char toUpperCase(char)
static char toLowerCase(char)
static char toTitleCase(char)

Void Class: - This class doesn't wrap any primitive value. It only represents the primitive type void.

Byte, Short, Integer, Long, Float and Double are inherited from Number class.

Math Class

- It is a final class.
- All methods are static methods.
- Math.E represent (log e) value
- Math.PI represents pi value

Round Methods

Static xxx abs(xxx) == returns the absolute value of the parameter
Static xxx min(xxx, xxx) == return the min value

Static xxx max(xxx, xxx) == returns the max value
Static double ceil(double) == returns the smallest double value which is !< argument.
Static double floor(double) == returns the largest double value which is !> argument.
Static int round(float) == rounds the value
Static long round(double) == rounds the value

Exponential Methods

Static double pow(double, double)
Static double exp(double)
Static double log(double)
Static double sqrt(double) == for –ve number result is NaN

Trigonometry Methods

Static double sin(double angle)
Static double cos(double angle)
Static double tan(double angle)

Random Methods

Static double random() == return the random number greater or equals to zero.

- What is the return type of hashCode() in Object class?

Select the right answer

a) String
b) int
c) long
d) Object
e) Class

Answer: - b

- Which of these statements are true?

Select all valid answers

a) if the references x and y are denote two different objects, then the expression x.equals(y) is always return false.
b) if the references x and y are denote two different objects then, the expression (x.hashCode() == y.hasCode()) is always return false.
c) The hashCode() in Object class is declared as final
d) The equals() in Object class is declared as final
e) All Array Objects have a method named clone.

Answer: - e [e is answer because it is extended from Object]

- Which of these exceptions can the wait() of Object class throw?

Select all valid answers

a) InterruptException
b) IllegalStateException
c) IllegalAccessException
d) IllegalThreadStateException

Answer: - a

- Which of the following are wrapper classes?

Select all valid answers

a) java.lang.Void
b) java.lang.Int
c) java.lang.Boolean
d) java.lang.Long
e) java.lang.String

Answer: - a, c, d

- Which of the following class doesn't extend java.lang.Number class?

Select all valid answers:

a) java.lang.Float
b) java.lang.Byte
c) java.lang.Character
d) java.lang.Boolean
e) java.lang.Short

Answer: - c, d

- Which of these wrapper classes produce immutable objects?

Select all valid answers:

a) Character
b) Byte
c) Short
d) Boolean

Answer: - a, b, c, d

- Given the following program, which of the lines print exactly 11?

```
public class MyClass extends Thread
{
        public static void main(String[] args)
        {
                double v = 10.5;
                System.out.println( Math.ceil(v));          //(1)
                System.out.println( Math.round(v));         //(2)
                System.out.println( Math.floor(v));         //(3)
                System.out.println( (int) Math.ceil(v));    //(4)
                System.out.println( (int) Math.floor(v));   //(5)
        }
}
```

Select all valid answers

a) The Line Labeled (1)
b) The Line Labeled (2)
c) The Line Labeled (3)
d) The Line Labeled (4)
e) The Line Labeled (5)

Answer: - b, d

- Which are the following methods are not a part of Math class?

Select all valid answers

a) double tan2(double)
b) double cos(double)
c) int abs(int)
d) double ceil(double)
e) float max(float,float)

Answer: - a

- what is the return type of round(float) of Math class?

Select the one right answer

a) int
b) float
c) double
d) Integer
e) Float

Answer: - a

- What is the return type of ceil(double) of Math class?

Select the one right answer

a) int
b) float
c) double
d) Integer
e) Double

Answer: - c

String Class

- It implements the immutable character strings, which are read only once string is created and initialized. For all successive operations new object will be created.
- Characters in string represented using the Unicode.

Creating and initializing Strings

- Create the string using the String lateral. Ex: - String str1= “welcome”; String lateral is implemented an anonymous String Object. Only one anonymous String Object shared by all string laterals which have the same content.
- New String(String s)

```
class Auxilitary
{
        public static String str1 = "WelCome";
}
public class MyClass extends Thread
{
        static String str1 = "WelCome";

        public static void main(String[] args)
        {
                String emptyStr = new String();
                System.out.println( "0: " + emptyStr);

                String str2 = "WelCome";
                String str3 = new String(str2);

                System.out.println( "1: "+ (str1 == str2) );
                System.out.println( "2: "+ str1.equals(str2) );

                System.out.println( "3: "+ (str2 == str3) );
                System.out.println( "4: "+ str2.equals(str3) );
```

```
                System.out.println( "5: "+ (str1 == Auxilitary.str1) );
                System.out.println( "6: "+ str1.equals(Auxilitary.str1) );
        }
}
```

Output: -
0:
1: true
2: true
3: false
4: true
5: true
6: true

Reading Individual characters in a String

- int length() == returns the length of the string. Length is an attribute to array Object. Length() is a method in String class.
- char charAt(int index) == returns the character at the given index in the string. If index is not valid then it throws StringIndexOutOfBoundsException.

String comparisons

- Characters are compared based on the Unicode values.
- Strings are compared lexicographically character by character.
- boolean equals(Object) == checking for equality of two strings.
- booelan equalsIgnoreCase(String) == checking for equality of 2 strings with out case of the characters.
- int compareTo(Object) == if object is String then it behaves as similar to below method else it throws ClassCastException.
- int compareTo(String) == Return 0 if both are equal; Return <0 if this string is lexicographically less than the argument; Return >0 if this string is lexicographically grater than the argument;

Character case in a String

- String toUpperCase()
- String toUpperCase(Locale)
- String toLowerCase()
- String toLowerCase(Locale)

```
public class MyClass extends Thread
{
        public static void main(String[] args)
        {
                String strA = new String("The Case was thrown out of Court");
                String strB = new String("the case was thrown out of court");

                String strC = strA.toLowerCase();
                String strD = strB.toLowerCase();

                System.out.println( "1: "+ (strC == strD) );
                System.out.println( "2: "+ (strC == strA) );
                System.out.println( "3: "+ (strD == strB)  );

                System.out.println( "4: "+ strC.equals(strD) );

        }
}
```

Output: -
1: false
2: false
3: true
4: true

[In above example, strA is modified and so new Object will be created with a reference strC. StrB is not modified even after toLowerCase() is applied on the it. So no new object is created. New reference is created for strB. So strB and strD are references to same object where as strA and strC are references to two different objects.]

Concatenation of Strings

- String concat(String)
- "+" used for Concatenation of two strings

Searching for characters and substrings

- int indexOf(int ch) == return the index of argument character string. Returns –1 if search is unsuccessful.
- int indexOf(int ch, int fromIndex)
- int indexOf(String)
- int indexOf(String, int fromIndex)
- int lastIndexOf(int ch) == return the index of argument character string from back wards. Returns –1 if search is unsuccessful.
- int lastIndexOf(int ch, int fromIndex)
- int lastIndexOf(String)
- int lastIndexOf(String, int fromIndex)
- String replace(char oldchar, char newchar)

Extracting substrings

- String trim() == remove the white space characters from both the ends
- String substring(int startIndex) == returns String starts at given index to end of the string.
- String substring(int startIndex , int endIndex) == returns the string starts at given index to endindex-1 of the String.

If indexes are invalid then StringIndexOutOfBoundsException will be thrown.

Conversion of Objects to Strings

- String toString() == returns the string representation of an object.
- Static String valueOf(Object)
- Static String valueOf(char[])
- Static String valueOf(xxxxx) == xxxx is a java data type

Other miscellaneous methods

- Char[] toCharArray()
- Byte[] getBytes()
- Boolean startsWith(String)
- Boolean endsWith(String)
- Int hasCode()

StringBuffer Class

- It implements the mutable character Strings.
- Not only the String buffer characters but also the StringBuffer Capacity can be changed dynamically.
- The capacity of StringBuffer is the no. Of characters that it can accommodate.
- String and StringBuffer are two independent Final classes. Both are extended from Object. Both are Thread safe.

Constructing the StringBuffers

- StringBuffer(String) == StringBuffer capacity = length of the string + 16 characters.
- StringBuffer(int length) == StringBuffer capacity = parameter. Parameter can't be < 0.
- StringBuffer() == StringBuffer capacity = 16 characters.

Changing and reading the StringBuffers

- int length() == return the no. Of characters in the String Buffer.
- Char charAt(int index) == read the character at specified index
- Void setCharAt(int index, char ch) == set the character at specified index.

Start index is 0. If index is invalid then it throws StringIndexOutOfBoundsException.

Construction strings from the StringBuffers

- String toString()

Appending, inserting and deleting characters form StringBuffers

- StringBuffer append(Object) == append at end of String buffer
- StringBuffer append(String)
- StringBuffer append(char[])
- StringBuffer append(char[] , int offset, int len)
- StringBuffer append(xxxxx) == xxx is java data type
- StringBuffer insert(int offset, Obejct o) == inserts the object at given position in stringBuffer
- StringBuffer insert(int offset, String s)
- StringBuffer insert(int offset, Char[])
- StringBuffer insert(int offset, xxxx) == xxx is java data type. Offset >= 0.
- StringBuffer deletecharAt(int index)
- StringBuffer delete(int startIndex, int endIndex) == inclusive start index and exclusive end index.
- StringBuffer reverse() == reverse the String Buffer contents

Controlling StrinBuffeı Capacity

- int capacity() == returns the capacity of the String buffer.
- Void ensureCapacity(int minCapacity) == it ensures that there is room for at least minCapacity characters in the string buffer if not it extends.
- Void setLength(int newLength) == ensures the actual no.of characters in the string buffer. Length of string buffer. NewLength >= 0. this operation results truncation (or) appending the null('\u0000') characters.

- Which of the following operators cannot be used in conjunction with a String Object?

Select all valid answers:

a) +
b) –
c) +=
d) .
e) &

Answer: - b, e

- Which one of these expressions will obtain the substring "kap" from a String defined by String str = "kakapo"?

Select the one right answer:

a) str.substring(2, 2)
b) str.substring(2, 3)
c) str.substring(2, 4)
d) str.substring(2, 5)
e) str.substring(3, 3)

Answer: - d

- What will be the result of attempting to compile and run the following code?

```
class MyClass
{
        public static void main(String[] args)
        {
                String str1 = "str1";
                String str2 = "str2";
                String str3 = "str3";
                str1.concat(str2);
                System.out.println( str3.concat(str1) );
        }
}
```

Select the one right answer:

a) The code will fail to compile, since str3.concat(str1) is not printable expression.
b) The program will print str3str2str1 when run.
c) The program will print str3 when run.
d) The program will print str3str1 when run.
e) The program will print str3str2 when run.

Answer: - d

- What function does the trim() method of the String Class perform?

Select the one right answer:

a) It returns a String where the leading white space of the original string has been removed.
b) It returns a String where the trailing white space of the original string has been removed.
c) It returns a String where the both leading and trailing white space of the original string has been removed.
d) It returns a String where the all the white space of the original string has been removed.
e) None of the Above

Answer: - c

- Which of the following statements are true?

Select all valid answers.

a) For any reference obj, the expression (obj instanceof obj) will return true.
b) You can make mutable subclasses of the String class.
c) All wrapper classes are declared as Final.
d) All objects have a public method named clone().
e) You can change the contents of a string object by using stringbuffer object.

Answer: - c

[a is wrong because instanceof operator LHS should be a reference and RHS should be a Class name. B is wrong because String is a Final class you can't make subclasses of String Class. D is wrong because clone() is protected method.]

- Which of these expressions are legal?

Select all valid answers:

a) "house".concat('boat")
b) ("house"+"boat")
c) (new String("house") + "boat")
d) ("house"+ new String("boat")

Answer: - a, b, c, d

- Which of these parameter lists have a corresponding constructor in the string class?

Select all valid answers:

a) ()
b) (int capacity)
c) (char[] data)
d) (string str)

Answer: - a, c, d

- Which of these methods is not part of the String class?

Select all valid answers

a) trim()
b) length()
c) concat(String)
d) hashcode()
e) reverse()

Answer: - e

- Which of these statements are concerning the charAt() of the String class are true?

Select all valid Answers:

a) The charAt() takes a char value as an argument
b) The charAt() returns a Character Object
c) The expression (“abcdef”).charAt(3) is illegal
d) The expression (“abcdef”).charAt(3) evaluates to the character ‘d’
e) The index of first character is 1

Answer: - d

- Which of these expressions will evaluate to true?

Select all valid answers

a) “hello: there!”.equals(“hello there”)
b) “HELLO THERE”.equals(“hello there”)
c) (“hello”.concat(“there”)).equals(“hello there”);
d) “Hello there”.compareTo(“hello there”) == 0
e) “Hello there”.toLowerCase().equals(“hello there”)

Answer: - e

- What is the result of attempting to compile and run the following program?

```
class MyClass
{
        public static void main(String[] args)
        {
                String s = "hello";
                StringBuffer sb = new StringBuffer(s);
                sb.reverse();
                if ( s == sb) System.out.println("a");
                if ( s.equals(sb)) System.out.println("b");
                if ( sb.equals(s)) System.out.println("c");
        }
```

}
Select the one right answer:

a) The code will fail to compile, since the constructor of the String Class is not properly called
b) The code will fail to compile, since (s == sb) is an illegal expression
c) The code will fail to compile, since the expression (s.equals(sb)) is illegal
d) The code will print c when run
e) The code will throw a ClassCastException when run

Answer:- b [String and StringBuffer are two different classes. There is no relation between them. Both are in compactable. So we will get compile time error.]

- What will be the result of attempting to compile and run the following program?

```
class MyClass
{
        public static void main(String[] args)
        {
                StringBuffer sb = new StringBuffer("have a nice day");
                sb.setLength(6);
                System.out.println(sb);
        }
}
```

Select the one right answer:

a) The code will fail to compile, since there is no method named setLength(0 in the StringBuffer class
b) The code will fail to compile, since sb is a StringBuffer reference, not a String reference and cannot be printed.
c) The program will throw a StringIndexOutOfBoundsException when run
d) The Program will print ‘have a nice’ day when run
e) The Program will print ‘have a’ when run
f) The Program will print ‘ce day’ when run

Answer: - e

- Which of these parameter lists have a corresponding constructor in the StringBuffer class?

Select all valid answers

a) ()
b) (int capacity)
c) (char[] data)
d) (String str)

Answer: - a, b, d

- Which of these methods is not part of the stringBuffer class?

Select all valid answers

a) trim()
b) length()
c) append(String)
d) reverse()
e) setLength(int)

Answer: - a

Collections Framework

- Collection allows a group of objects to be treated as a single unit. Arbitrary objects can be stored, retrieved and manipulated as elements of collection.
- Collections framework presents a set of utility classes to manage such collections.

Java.util package are comprises three main parts:

1. Interfaces
2. Set of implementations – well known data structures based on interfaces.
3. Assortment algorithms, which can perform sort and search on collections.

Interface	Description
Collection	A basic interface that defines operations that all the classes that maintain collections of objects typically implement.
Set	Extends Collection interface for sets maintain unique elements. Duplicates are not allowed.
SrotedSet	Extends Set interface for sets maintain their elements in sorted order.
List	Extends Collection interface for lists maintain their elements in a sequence. (Elements in order) need not be unique. It is also known as Sequence. It may contain duplicates.
Map	A basic interface that defines operations that classes represent mappings of keys to values typically implement.
SortedMAp	Extends the Map interface for maps that maintain their mappings in key order.

Data Structures	Set Interface Impl	SortedSet Interface Impl	List Interface Impl	Map Interface Impl	SortedMap Interface Impl
Hash table	HashSet class			HashMap, Hashtable classes	
Resizable Array			ArrayList, Vector Classes		
Balanced Tree		TreeSet class			TreeMap class
Linked List			LinkedList class		

- Collections and Map's are not interchangeable.
- **Java.util.Collections** provides static methods, which implement polymorphic algorithms for various operations (sorting, searching, shuffling etc.) on Collection Objects. It has useful factory methods for creating Collection instances.
- Some of the operations on Collection interface are optional, meaning that a collection may not choose to provide a proper implementation of such operation. In such case an **UnsupportedOpeartionException** is thrown when optional operation is invoked.

Collection interface Operations

- int size()
- boolean isEmpty()
- boolean contains(Object element)
- boolean add(Object element) ----------→ Optional
- boolean remove(Object element) ----------→ Optional

- boolean containsAll(Collection c) ----------→ Optional[works as SUBSET of collections]
- boolean addAll(Collection c) ----------→ Optional [works as UNIOUN of collections]
- boolean removeAll(Collection c) ----------→ Optional[works as DIFFERENCE of collections]
- boolean retainAll(Collection c) ----------→ Optional[works as INTERSECTION of collections]
- void clear() ----------→ Optional

Array Operations

- Object[] toArray() == fills an array with elements of collection
- Object[] toArray(Object[] a) == it used to specify the type of array into which the collection are to be stored. If array is big enough then elements are stored in this array. If there is a room to spare in the array, the spare room is filled with nulls before array is returned. If array is not enough for collection then a new array of same type will be create with appropriate size and returned.

Iterators: - it allows serial access to the elements of a collection. It is an interface.

```
interface Iterator
{
        boolean hasNext();
        Object next();
        void  remove(); ----------→ Optional
}
```

Remove() is the only recommended way to remove elements from a collection during the iteration.

Set

HashSet implemnts Set interface

TreeSet implements SortedSet interface

Initial capacity – No. Of buckets in the hash table.
Load Factor – The ratio of number of elements stored to its current capacity.

- HashSet()
- HashSet(int initialCapacity)
- HashSet(int initialCapacity, int loadFactor)

List

It defines some operations, which are especially for LIST type Collections.

- Object get(int index) [get the element at specified index]
- Object set(int Index, Object element) --------> Optional[replace element at specified index]
- Void add(int Index, Object element) --------> Optional[inherited method add() adds the element at end side of list where as this method will add the element at specified index.]
- Object remove(int Index) -----> Optional[inherited method remove() removes the element from first occurrence where as this method will remove the element at specified index.]
- Boolean addAll(int Index, Collection c) -------> Optional[add all the elements from specified collection at the specified index using the collection's specified iterator.]
- int indexOf(Obejct o) [returns the index of first occurrence of the Object else return -1]
- int lastIndexOf(Obejct o) [returns the index of first occurrence of the Object else return -1]
- ListIterator listIterator() [it starts from the starting point of iterator.]
- ListIterator listIterator(int startIndex) [it starts from the specified index of iterator.]

First element index is 0(zero) last element index is size()-1.
In valid index throws IndexOutOfBoundsException.

ListIterator: - customized iterator for lists.

```
Interface ListIterator extends Iterator
{
        Boolean hasNext();
        Boolean hasPrevious();

        Object next();
        Obejct previous();

        int nextIndex();
        int previousIndex();

        void remove(); ----------------------> Optional
        void set(Object); -------------------> Optional
        void add(Object); -------------------> Optional

        List subList(int fromIndex, int toIndex); [ Returns a sublist from specified inclusive start index and exclusive end index.]
}
```

ArrayList, Vector, LinkedList

- All three are the implementations of List interface.
- All three provides a standard constructor to create empty list.
- All three provides a standard constructor for list based on the existing collection.
- ArrayList and Vector allows creating the empty list with an initial capacity.
- ArrayList and Vector are dynamically resizable arrays.
- **Vector class is Thread-safe. ArrayList is not thread-safe.**
- ArrayList performance is comparably better than vector but it doesn't have synchronization. But positional access has constant time in both is same.
- In most cases implementation of ArrayList is best choice for Lists. Where frequent insertions and deletions are occurred inside list then LinkedList is worth of choice.

Maps

- Defines a mapping from keys to values
- Doesn't allow duplicate keys
- Each key map at most one value
- It is not a Collection.
- It also has some optional methods. When user invoked the optional methods in unimplemented class then it throws unSupportedOperationException.
- Object put(Object key, Object val); ----------------> Optional
- Object get(Object key);
- Object remove(Object key); ----------------> Optional
- Boolean containsKey(Object key);
- Boolean containsValue(Object val);
- Int size();
- Boolean isEmpty();
- Void putAll(Map t); ----> Optional
- Void clear(); ----> Optional
- Set keySet();
- Collection values(); [returned Collection is not a set. Collection also contains duplicates.]

- Set entrySet();

```
Interface Entry{
        Object getKey();
        Object getValue();
        Object setValue(Object val);
}
```

HashMap & Hashtable

- These 2 implement the Map interface.
- Constructors are available to create empty map.
- Constructors also available to create empty map based on the Collection.
- Constructors are available to create map with initial capacity and load factor.
- **Hashtable is thread-safe. HashMap is not thread-safe.**

Interface comparator

```
{
        int compare(Obejct o1, Object o2);
}
```

compare() returns negative number, zero and positive number if the current object is less than, equal , greater than the specified Object, according to the natural order. Since this is also checks for equality of the Objects but its implementation should not be contradict with equals ().

- All wrapper classes, String, Date, File class's implements this interface.
- All wrapper classes, String classes are final classes.
- **Objects implementing comparator interface can be used** 1) as elements in the SortedSet 2) as keys in the SortedMap 3) in lists which can be sorted automatically by Collections.sort().

SortedSet interface extends Set interface to provide the functionality for handling sorted Sets.

- SortedSet headSet(Object toElement) – returns the subset of Set whose elements are strictly less than the specified element.
- SortedSet tailSet(Object fromElement) – returns the subset of Set whose elements are greater than or equals to the specified element.
- SortedSet subSet(Object fromElement, Object toElement) – returns the subset of Set whose elements are lying between the from and to elements. FromElement is inclusive and toElement is exclusive.
- Object first() – returns the first(minimum) element in the SortedSet.
- Object last() – returns the last(Maximum) element in the SortedSet.
- Comparator comparator() – returns the comparator associated with this SortedSet (or) null if it uses the natural ordering of its elements.

SortedMap interface extends Map interface to provide the functionality for implementing Maps with Sorted Keys.

- SortedMap headMap(Object toKey) – returns the subset of Map whose Keys are strictly less than the specified Key.
- SortedMap tailMap(Object fromKey) – returns the subset of Map whose Keys are greater than or equals to the specified Key.
- SortedMap subMap(Object fromKey, Object toKey) – returns the subset of Map whose Keys are lying between the from and to Keys. FromKey is inclusive and toKey is exclusive.
- Object firstKey() – returns the first(minimum) Key in the SortedMap.
- Object lastKey() – returns the last(Maximum) Key in the SortedMap.
- Comparator comparator() – returns the comparator associated with this SortedMap (or) null if it uses the natural ordering of its Keys.

TreeSet implements SortedSet interface & TreeMap implements SortedMap interface

- Constructors are as follows:

TreeSet()	TreeMap()
TreeSet(Comparator)	TreeMap(Comparator)
TreeSet(Collection)	TreeMap(Map)
TreeSet(SortedSet)	TreeMap(SortedMap)

Customizing the Collections

Collection implementations can be customized with regards of 2 things.

1. Thread-safety – Vector & hashtable are default thread-safe
2. Data immutability

A decorator Object "wraps around" a Collection and modifies the behavior of Collection. Java provides static factory methods, which return appropriately decorated collection instances. These are known as "anonymous implementations".

Synchronized Collection Decorators

(All methods are static methods of java.util.Collections class)

- Collection synchronizedCollection(Collection c)
- List synchronizedList(List li)
- Set synchronizedSet(Set s)
- Map synchronizedMap(Map m)
- SortedSet synchronizedSortedSet(SortedSet ss)
- SortedMap synchronizedSortedMap(SortedMap sm)

Un modifiable Collection Decorators (Read only access on the Collections).

(All methods are static methods of java.util.Collections class)

- Collection unmodifiableCollection(Collection c)
- List unmodifiableList(List li)
- Set unmodifiableSet(Set s)
- Map unmodifiableMap(Map m)
- SortedSet unmodifiableSortedSet(SortedSet ss)
- SortedMap unmodifiableSortedMap(SortedMap sm)

Any operation, which modifies the collection, will give unSupportedOperationException.

- **Singleton** (a set containing only one element)[static methods of java.util.Collections class]
- public static Set **singleton**(Object o)
- public static List **singletonList**(Object o)
- public static Map **singletonMap**(Object key, Object value)
- public static List **nCopies**(int n, Object o) -- Returns an immutable list consisting of n copies of the specified object. The newly allocated data object is tiny (it contains a single reference to the data object)
- asList() of Array class and toArray() of Collection class matain a bridge between the Arrays and Collections.

Abstract Implementations

These are abstract classes only.

Ex:- interface Set extends Collection interface.
AbstractSet class implements Set, Collection interfaces extends AbstractCollection class.
HashSet extends AbstractSet class.

- Which of these are core interfaces in the Collections framework?

Select all valid answers.

a) Set
b) Bag
c) LinkedList
d) Collection
e) Map

Answer: - a, d, e

- Which of these implementations are provided by java.util package?

Select all valid Answers:

a) HashList
b) HashMap
c) ArraySet
d) ArrayMap
e) TreeMap

Answer: - b, e

- Which is the name of the interface used to represent Collections that maintain non-unique elements in order?

Select all valid answers:

a) Collection
b) Set
c) SortedSet
d) List
e) Sequence

Answer: - d

- Which of these statements concerning the use of Collection operations are true?

Select all valid answers:

a) Some operations may throw UnsupportedOperationExcetption.
b) Methods using some operations must either catch unsupportedOperationException or declare that they throw such exceptions.
c) Collection classes implementing List can have duplicate elements.
d) ArrayList can only accommodate fixed no. Of elements
e) The Collection interface contains a method named get.

Answer: - a, c

- Which of these are methods are defined in the Collection interface?

Select all valid answers:

a) add(Object o)
b) retainAll(Collection c)
c) get(int Index)
d) iterator()
e) indexOf(Obejct o)

Answer: - a, b, d

- Which of these methods from the collection interface return the value true if the collection object was modified during the operation?

Select all valid answers:

a) contains()
b) add()
c) containsAll()
d) retainAll()
e) clear()

Answer: - b, d

- What will be the result of attempting to compile and run the following code?

```
import java.util.*;
public class MyClass
{
	public static void main(String[] args)
	{
		HashSet set1 = new HashSet();
		addRange(set1,1);
		ArrayList list1 = new ArrayList();
		addRange(list1,2);
		TreeSet set2 = new TreeSet();
		addRange(set2,3);
		LinkedList list2 =  new LinkedList();
		addRange(list2,5);

		set1.removeAll(list1);
		list1.addAll(set2);
		list2.addAll(list1);
		set1.removeAll(list2);

		System.out.println( set1 );
	}
	static void addRange(Collection col, int step)
	{
		for ( int i = step * 2; i <= 25 ; i+=step )
		{
			col.add(new Integer(i));
		}
	}
}
```

Select one right answer:

a) The program will fail to compile since operations are performed on mismatching collection implementations.
b) The program will fail to compile since the TreeSet instance has not been given a comparator to use when sorting its elements.
c) The program will compile without error, but will throw UnsupportedOperationException when run
d) The program will compile without error, but will print all primes below 25 when run
e) The program will compile without error, but will print some other sequence of numbers when run

Answer: - d

- Which of these collections implementations are thread-safe?

Select all valid answers:

a) ArrayList
b) HashSet
c) Vector
d) TreeSet
e) LinkedList

Answer: - c

- Which of these methods can be called on the objects implementing the Map interface?

Select all valid answers:

a) contains(Object o)
b) addAll(Collection c)
c) remove(Object o)
d) values()
e) toArray()

Answer: - c, d

- Which of these statements are concerning maps are true?

Select all valid answers:

a) The return type of the values() method is Set
b) Changes made to the Set view returned by keySet() will be reflected in the Original Map.
c) The Map interface extends Collection interface
d) All Keys are unique in Map
e) All Map implementations keep keys sorted.

Answers: - b, d

- Which of these classes has a comparator () method?

Select all valid answers:

a) ArrayList
b) HashMap
c) TreeSet
d) HashSet
e) TreeMap

Answer: - c, e

- Which sequence of digits will the following program print?

```
import java.util.*;
public class MyClass
{
        public static void main(String[] args)
        {
                List list = new ArrayList();
                list.add("1");
                list.add("2");
                list.add(1,"3");
                List list2 = new LinkedList(list);
                list.addAll(list2);
```

```
                list2 = list.subList(2,5);
                list2.clear();

                System.out.println( list );
        }
}
```

Select the right answer:

a) The sequence 1,3,2 is printed
b) The sequence 1,3,3,2 is printed
c) The sequence 1,3,2, 1, 3, 2 is printed
d) The sequence 3,1,2 is printed
e) The sequence 3,1,1,2 is printed
f) None of the above

Answer: - a [sub List is created from List, operations performed on the sub list will reflect in the main List.]

- What will be the output of the following program?

```
class Base {
        Base()
        {
                System.out.println("parameterless Base");
        }
        Base(int i)
        {
                System.out.println("with parameter Base");
        }
}
class Derived extends Base {
        int ij = 10;
        Derived()
        {
                ij = 20;
                System.out.println("ij = " + ij);
                new Derived(5);
                System.out.println("parameterless Derived");
                System.out.println("ij = " + ij);
        }
        Derived(int i)
        {
                ij = 30;
                System.out.println("with parameter Derived");
                System.out.println("ij = " + ij);
        }
        public static void main(String args[])
        {
                int i = 5;
                Base b = new Derived();
        }
}
```

Answer: -

```
parameterless Base
ij = 20
parameterless Base
with parameter Derived
ij = 30
parameterless Derived
ij = 20
```

- Why do you use Final variables?

Answer: - Final variables are constants throughout the application. Final variables optimize the performance.

- What will be the output of the following program?

```
class Base {
	Base()
	{
		System.out.println("parameterless Base");
	}
	Base(int i)
	{
		System.out.println("with parameter Base");
	}
}
class Derived extends Base {
	Derived()
	{
		new Derived(5);
		System.out.println("parameterless Derived");
	}
	Derived(int i)
	{
		System.out.println("with parameter Derived");
	}
	public static void main(String args[])
	{
		int i = 5;
		Base b = new Derived();
	}
}
```

output: -

```
parameterless Base
parameterless Base
with parameter Derived
parameterless Derived
```

- **What is the difference between the AWT and SWING component?**

SWING	AWT
➢ SWING is not based on PEER	➢ AWT is based on PEER.
➢ SWING is platform independent wherever JVM is available.	➢ PEER is associated with every component. PEER is created when component is created. The responsibility of PEER is communication with under laying operating system. The Operating system is responsible for Rendering (appearance) of a component and interaction with the user. PEER interacts with Operating system using the native libraries. Native libraries are depending upon the operating system. So, PEER is platform dependent. So AWT is platform dependent.
➢ Architecture of SWING is MVC. ➢ Model: - Storing the State ➢ View: - appearance ➢ Controller: - user interface	➢ Architecture of AWT is ➢ Component = PEER = Operating system
➢ Same model have multiple views depends upon the requirement.	➢ Multiple views are not possible for a component.
➢ Operating system provides the rendering region for a component.	➢ Operating system provides user interface, appearance and rendering region for a component.
➢ Uniform appearance over multiple platforms.	➢ It is not possible.
➢ SWING has pluggable look and feel. Default is java look and feel.	➢ It is not pluggable.
➢ SWING components are lightweight components.	➢ AWT components are heavyweight components.