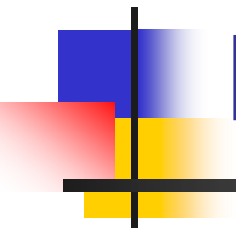
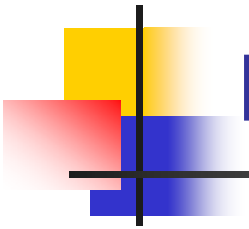


Block I – Unit 2

Basic Constructs in Java



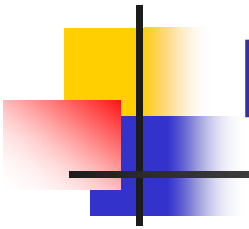
Developing a Simple Java Program

- Objectives:
  - Create a simple object using a constructor.
  - Create and display a window frame.
  - Paint a message in a window.
  - Use data fields to store simple values.
  - Examine the class *"Thread"* and some of its features.
  - Refer to a superclass using the reserved word *"super".*
  - Build a simple animation using a loop.

Achieving the objectives, by modifying FirstWorld

- Most of the Objectives can be tested separately throughout Practical Activities 1.1 → 1.3

- Let's take a look (Load FirstWorld project and Run Application).



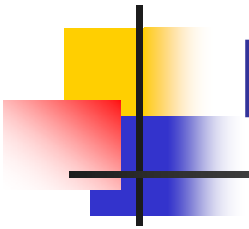
Practical Activity 1.4

- The reserved word **super** refers to the superclass of the class in which it is used.
  - For example: FirstWorld extends JFrame and hence we can invoke methods of JFrame by using the reserved word "super".
  - Check out FirstWorld

Practical Activity 1.5

- The purpose of this activity is to introduce you to some simple animation using a loop.
- Modify the run method of FirstWorld as follows and examine its behavior:

```
Public void run() {
   for (int i = 0; i < 10; i++){
      try{
         Thread.sleep(1000);
      }catch (Exception e) {System.exit(0);}
      xCoord = xCoord + 40;
      yCoord = yCoord + 25;
      repaint();
   }
}
```



Practical Activity 1.6

- Animate a graphical **object** rather than a text object.

- Probably one of the easiest ways to create a graphical object is to make use of the awt class "*Graphics*".

Practical Activity 1.6

Some *Graphics* Methods	What it is used for
drawString	Draws some text
drawLine	Draws a line
drawOval	Draws an empty Oval
fillOval	Draws an Oval and fills it using the current Color
fillRect	Draw a Rectangle and fills it using the current Color

Practical Activity 1.6

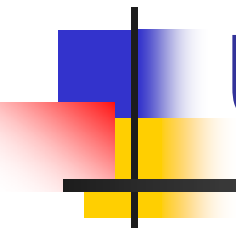
- Let us look at Disk.java
- Load FirstWorldGraphics and run it.

(Practical Activity 1.7) How can you make the disk moves more smoothly ?!

Unit 2 – Section 1 - Key Terms

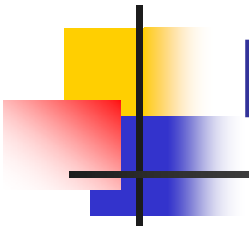
- Key Terms and Concepts to Remember:

actual argument	initialize	**public**
catch	interrupt	redefine (a method)
coordinate	invoke (a method)	**repaint**
constructor	loop	**sleep**
data field	loop control variable	**static**
event model	**main**	subclass
final	**paint**	**super**
for	pixel	supeclass
graphics context	**Point**	**Thread**
inheritance	**private**	
try	**protected**	



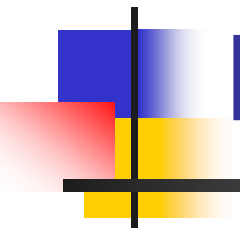
Unit 2

Section 2



Ball Worlds

- Objectives:
  - Create a small application that uses the AWT (Abstract Windowing Toolkit) Swing packages to simulate movement in a window based on the Java graphics model.
  - Develop Java programs with several classes using inheritance.
  - Define and use object constructors.
  - Define constants in java
  - Represent a simple java program as a class diagram
  - Construct a Java program by accessing code held in files.



Ball Worlds

Using BallWorld to understand a number of Java concepts

Let's take a look at the execution of BallWorld first!

The *new* Expression

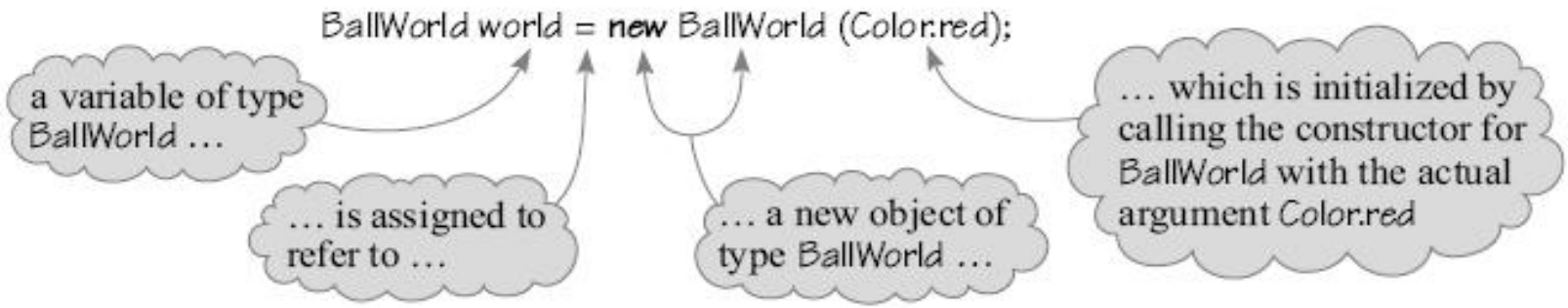


Figure 2.1

- Meaning of the above statement:
  - Create a new object of type BallWorld, initialized using the constructor, and assign it to the variable, world.
  - Let us take a look at BallWorld code

Calling methods and argument passing

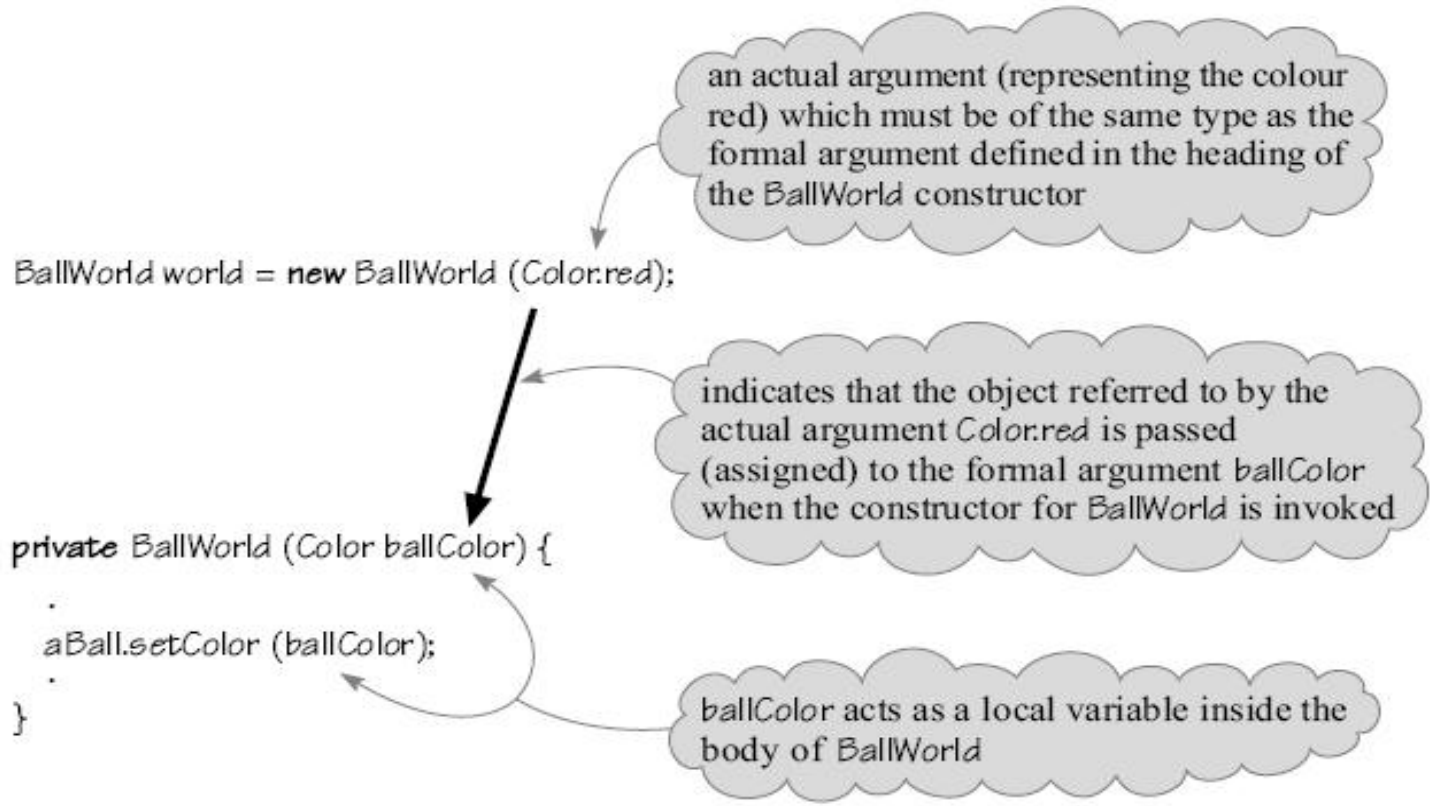


Figure 2.2 Argument passing

Calling methods and argument passing

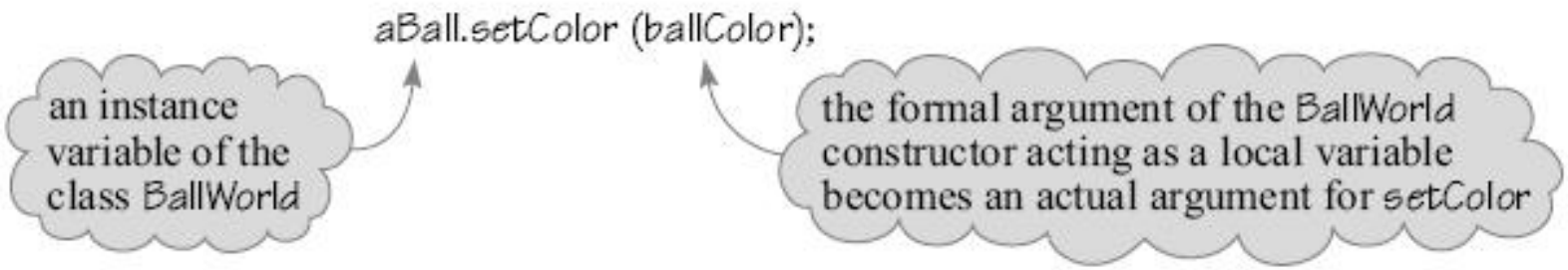


Figure 2.3 The value of ballColor is passed to setColor

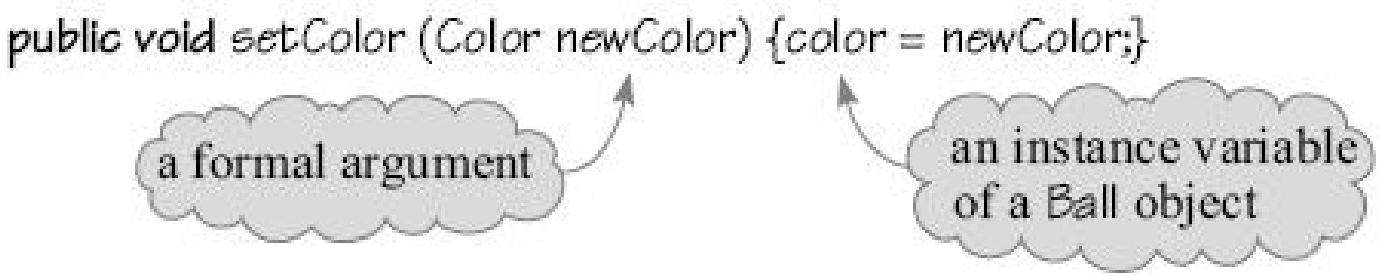


Figure 2.4 The setColor method

Objects and References

- To understand objects and references, we shall examine what happens to a ball object in the class BallWorld. The object is first defined in the following line:

```
private Ball aBall = new Ball (new Point(50, 50), 20);
```

Constructors used:

From class Ball:	public Ball (Point lc, int r) {loc = lc; rad = r;}
From class Point:	public Point (int x, int y)

Objects and References

- At this stage aBall can be pictured as follows:

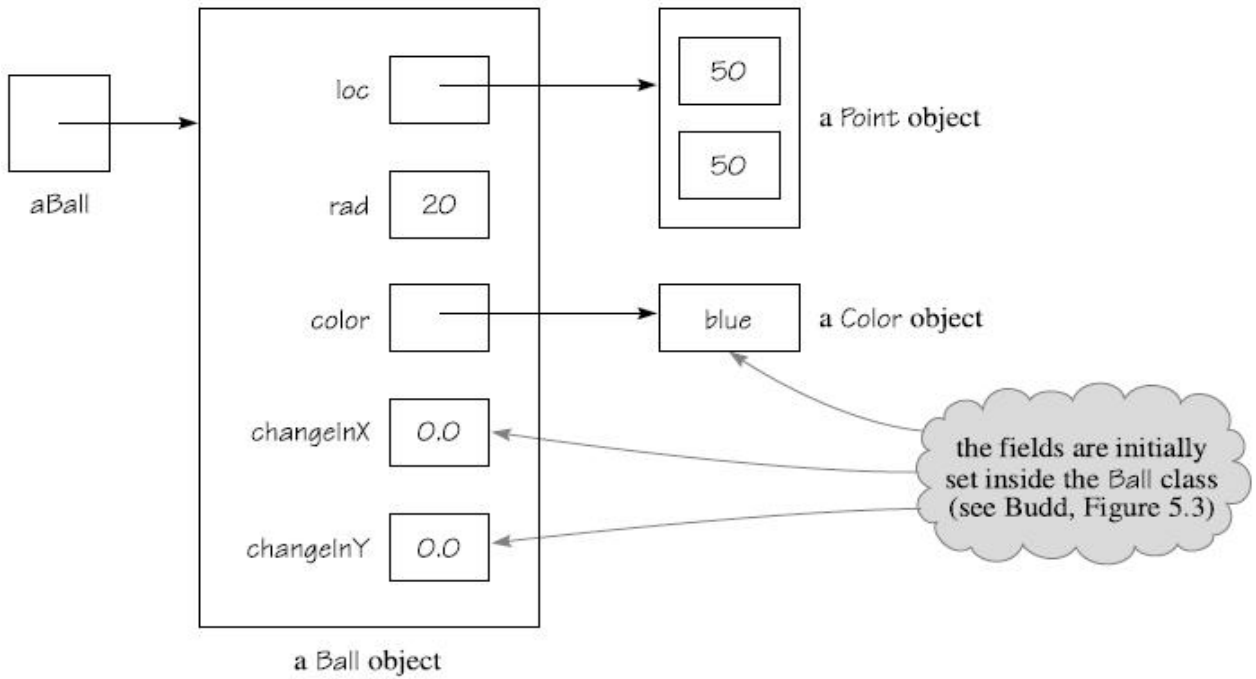


Figure 2.5 An initialized Ball object

Ball Source Code

Objects and References

Inside the constructor for BallWorld the ball referenced by aBall is subject to the following two method calls:

aBall.setColor (ballColor);

aBall.setMotion (3.0, 6.0);

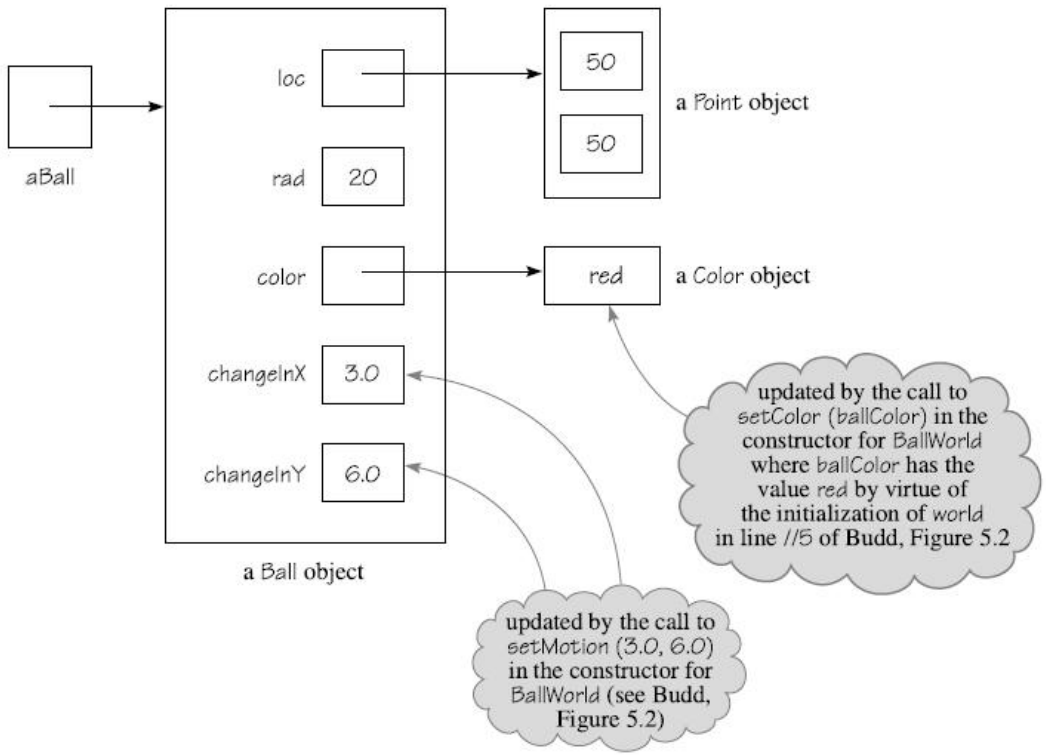


Figure 2.6 The Ball object after two setter messages

BallWorld Source Code

Ball Source Code

Objects and References

- What happens when we execute: aBall.move();
- First the following method from class Ball is executed:

```
public void move()
    {
        loc.translate((int) changeInX, (int) changeInY);
    }
```

- Which in turn executes the following method from class Point:

```
public void translate (int dx, int dy)

// Translates the Point at location (x, y) by dx along the x axis
// and dy along the y axis.
```

Objects and References

- The result of the execution is the following:

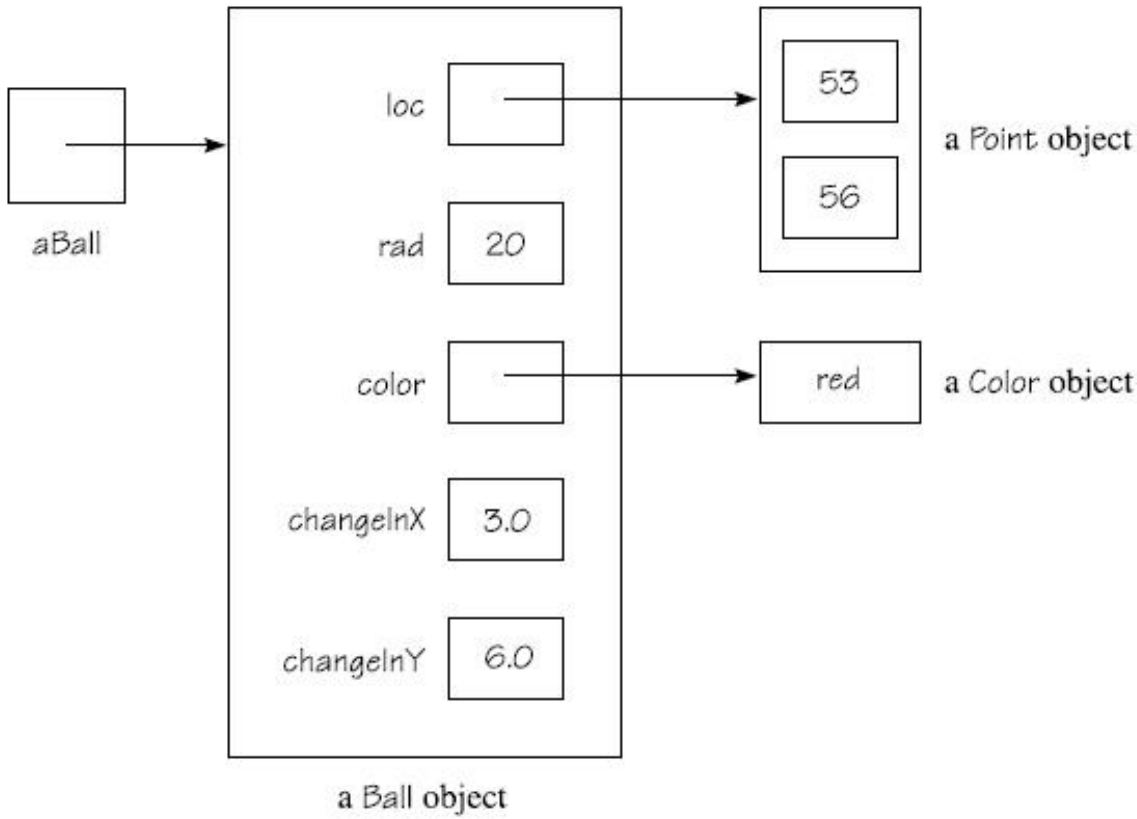
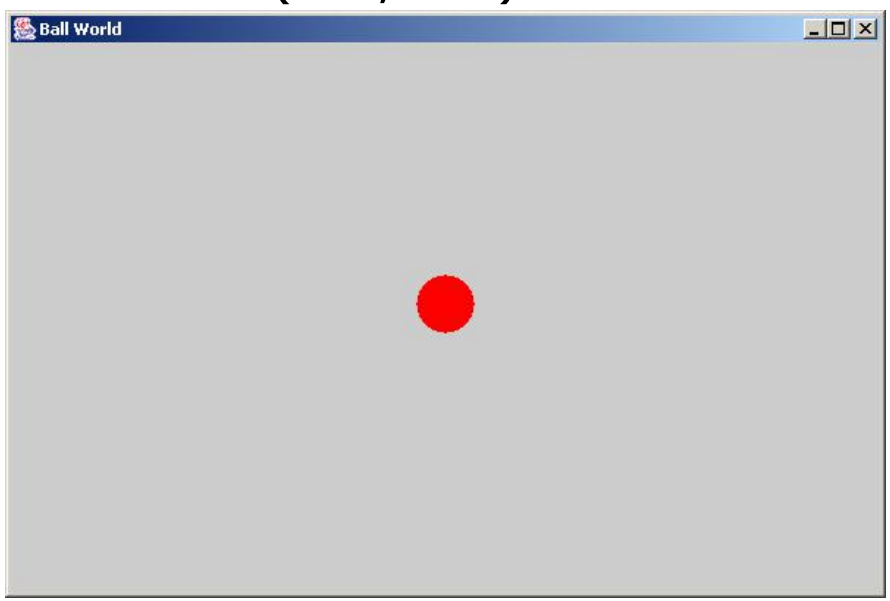
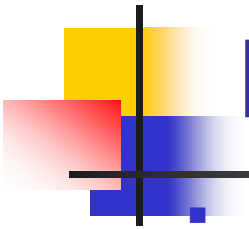


Figure 2.7 A Ball object

Java Windows

- In the BallWorld class, the window has a width of 600 and a height of 400 pixels. This means that any point within the boundary of the window must have an x-value between 0 and 600, and a y-value between 0 and 400. Therefore, the point in the middle of the window will have coordinates (300, 200).





Receiver Object

- BallWorld inherits from JFrame
- JFrame inherits from Frame that has the following 2 methods:

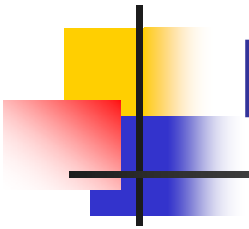
setSize (int width, int height)
setTitle (String title)

Note: setSize is originally inherited from class Component

- Both methods were called from BallWorld without a receiver object. Java allows such an implementation where there is no confusion.

- However, Java has a reserved word "*this*" which can be used to denote the current object where lines 20 and 21 of BallWorld.java could have been written as follows:

this.setSize (FrameWidth, FrameHeight);
this.setTitle ("Ball World");



Rectangles, Ovals and Balls

- In Java, the easiest way to draw solid objects is to use the Graphics class. One of its methods is fillOval which takes 4 arguments which are:
  - The 2 coordinates of the top left-hand corner of the rectangle.
  - The width of the rectangle
  - The height of the rectangle

Rectangles, Ovals and Balls

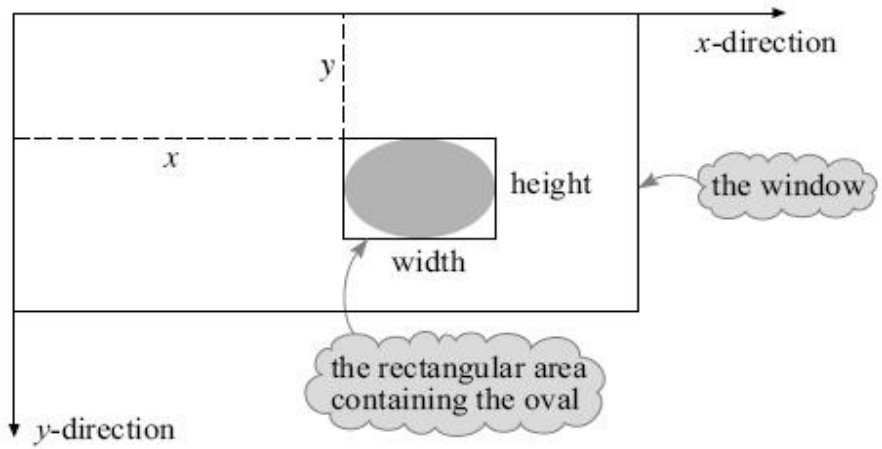


Figure 2.8 The arguments defining an oval shape in a window

- The Ball paint method contains the statement:
  g.fillOval (loc.x-rad, loc.y-rad, 2*rad, 2*rad);

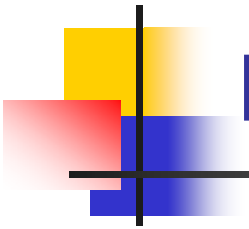
If (x, y) = (50, 50) and rad = 20, then the above method will create an Oval at location (30, 30) with width = 40 and Height = 40

Ball Source Code

Moving Objects on the Screen

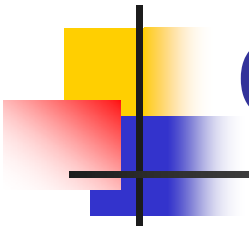
- Movement of Ball is obtained by continually:
  - Changing, by a small amount, the coordinates of the *Point loc* which defines the centre of the ball,
  - Repainting the window with the circle in its new position.

Ball Source Code



Moving windows about the screen

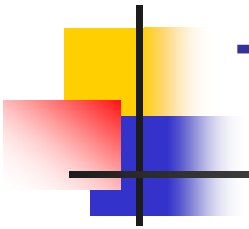
- When you run the BallWorld program you will have a moving ball bouncing inside a window.
- If you move the window, the bouncing ball will continue to move.
- However if you resize the window, the simulation does not automatically resize.
  - Let's take a look at the execution of BallWorld and examine the above scenarios



Graphics Objects

- A graphics object enables you to draw graphical objects on the screen and has a number of data fields, such as *color* , which specify how the graphical object is to appear (or be rendered) on the screen.

- For the present, simply accept that you require a graphics object as an argument to the *paint* method.



The paint and repaint methods

- Repainting the screen in Java is a two-stage process.
  - First, you call *repaint* and then
  - *repaint* calls *paint* for you.
- In fact, when you invoke the method *show* , in the *main* method, *show* actually calls *repaint* : you cannot call *paint* directly. So, when you wish to change the contents of a window, you should always call *repaint.*
- *paint* and *repaint* are an example of Java's event model that you will study later.

BallWorld Source Code

BallWorld flow of control

- When you execute the BallWorld application the following methods are executed in order:
  - main
  - BallWorld constructor
  - world.show
  - repaint and paint
  - world.run 1000 times
  - System.exit(0)

BallWorld Source Code

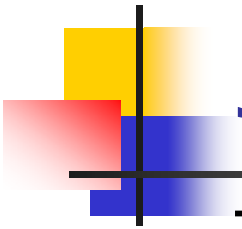
The operator ++

- In the *BallWorld Main* method, the following ***for*** statement occurs:

```
for (int i=0; i < 1000; i++)
```

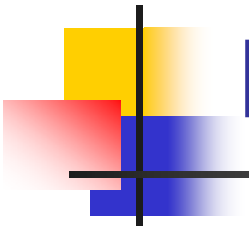
- The expression i++ is a shorthand for the assignment i = i + 1 .

BallWorld Source Code



Java 2

- Java is developed by Sun Microsystems
- Java 2 Platform is a term that indicates Java versions 1.2 and above.
- Those versions included a revised run-time library with a windowing interface that is more platform independent than the AWT.
- In practical terms, part of the AWT has been replaced by a new set of classes known as **Swing classes** .



Modeling

- Will be examined in more details in Blocks 4 and 5.
- Design is aimed at providing more detail on how the application classes are to be constructed
- and, in particular, on the components (classes from packages, in the case of Java) that can be used in their construction.

Modeling

- In anticipation of the work on analysis and design you will be required to do in Blocks 4 and 5, a class diagram of the *BallWorld* program is shown below

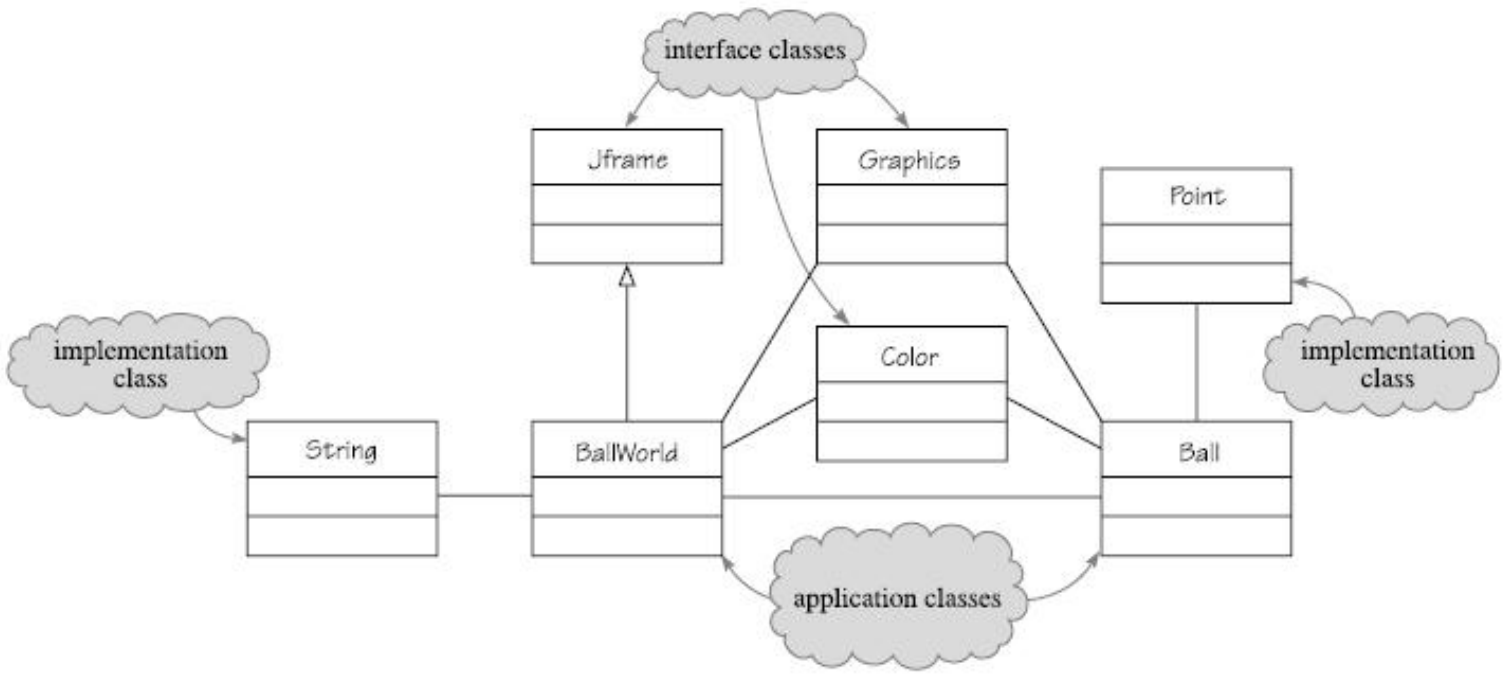
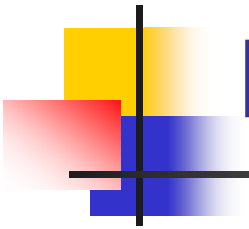


Figure 2.9 The class diagram for the BallWorld program



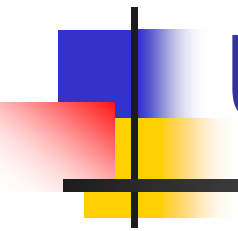
Modeling

- 2 main types of relationship can exist between classes:
  - **generalization** (that is, inheritance). For example, *BallWorld* inherits from *JFrame*
  - **association** which involves one class having one or more variables of the other class in it. For example *BallWorld* is associated with *Graphics* because of the use of the *Graphics* argument *g* in its *paint* method.

Unit 2 – Section 2 - Key Terms

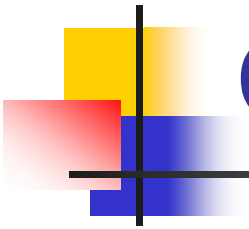
- Key Terms and Concepts to Remember:

accessor methods	debugging	interfce class
actual argument	formal argument	**new**
application class	frame (window)	object initialization
argument passing	framework	object creation
class diagram	graphics model	package
constants	implementation class	swing classes
constructor	inheritance	**this**



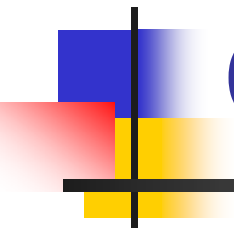
Unit 2

Section 3



Cannon Game

- Objectives:
  - Understand the need for the **'wrapper'** class Integer
  - Use **inheritance** to create a new class based on Ball class introduced in Section 1
  - Use the pseudovariable **super**
  - Explain the use of **inner classes** and explain why they are useful
  - Describe Java's **event model** (events, listeners, buttons, scrollbars)
  - Describe and use the default window **layout manager**
  - Write code which enables a window application in Java to be terminated by closing its window.



Cannon Game

Using CannonGame to understand more about windowing facilities and user interface components in Java

Let's take a look at the execution of CannonGame first!

The Value *null*

- The statement:

  private Ball aBall = null;

  - Introduces an instance variable, aBall, that will subsequently be used to refer to an object of type CannonBall.
  - As this object has not yet been created, there is nothing for aBall to refer to.

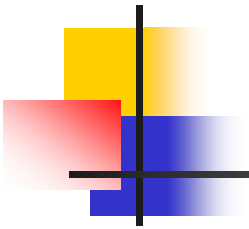
CannonGame Source Code

The modifier for class *Cannon*

- The fact that there are no modifiers preceding the class name *Cannon* means that, by default, the class is only accessible to other classes in the same package.

Cannon Source Code

The class *Integer* and the primitive data type *int*



- *int*
  - a primitive data type .
  - its operations are not represented by methods and are confined to the normal arithmetic operations such as add and multiply.
- *Integer*
  - an Object which contains an *int* value that can be manipulated by methods.
  - Instances of the class *Integer* are objects such that each *Integer* object contains (or 'wraps up') a data field of type *int*.

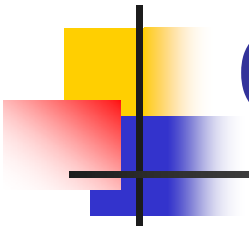
The class *Integer* and the primitive data type *int*

- Some useful methods of class *Integer:*
  - valueOf
    - Example: Integer.valueOf("5") converts the String "5" into an Integer with value 5.
  - intValue
    - Example: Assume we have an Integer object (Int1) with value 6, the method Int1.intValue() returns the primitive data type *int* of value 6.
  - You can also convert a value of type *int* (such as 42) into an *Integer* object by writing,
    - Integer (42)

Radians

- Budd uses Radians to set the angle measurement of the Cannon.
- As stated in the Glossary:
  - Radian: A measure of angularity. There are 2Π (two pi) radians in 360º and therefore 1º = Π/180 radians. One radian equals approximately 57º

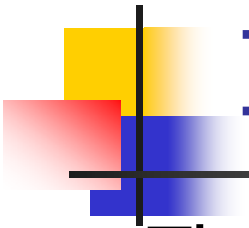
Cannon Source Code



Casts

- In the paint method of class Cannon there are the following expressions:
  - int lv = (int) (barrelLength * Math.sin(radianAngle));
  - int lh = (int) (barrelLength * Math.cos(radianAngle));
  - int sh = (int) (barrelWidth * Math.sin(radianAngle));
  - int sv = (int) (barrelWidth * Math.cos(radianAngle));

- radianAngle is of type *double and* (barrelLength * Math.sin(radianAngle)) is also of type *double*
- Putting *(int)* in front of a *double* simply changes the value to an integer by ignoring anything that comes after the decimal. This is usually known as "Casting".



Inner Classes

- These are commonly used in Java to handle events.
- Java has three types of inner class, but, in this course, you will be introduced to two of them:
  - member classes, where the inner class is defined at the same level as other members (data fields and methods) of the class
  - unnamed classes, which will be examined later in Unit 5

CannonWorld Source Code

Constructors for Listener Objects

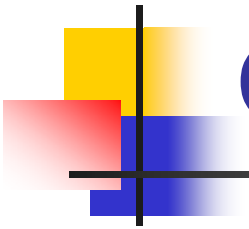
```
private class FireButtonListener implements ActionListener{
   public void actionPerformed(ActionEvent evt){
     aBall = cannon.fire();
   }
 }
```

- The Class above has no Constructor, even though there is the following line of code in the program:

new FireButtonListener()

CannonWorld Source Code

Is that possible? →

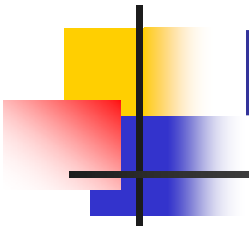


Constructors for Listener Objects

- The statement

  new FireButtonListener()

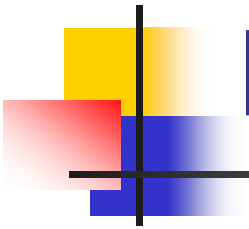
- Refers to a **default constructor** which is a constructor already known to the Java system and has no arguments.
- It will be used when a class is defined without its own constructor.
- Such type of constructors create a new object of the class and does nothing more.



Events

```
private class FireButtonListener implements ActionListener{
    public void actionPerformed(ActionEvent evt){
      aBall = cannon.fire();
    }
  }
```

- ActionListener is a class that actually listens (waits) for an action.
- actionPerformed is a method that is invoked whenever an action happens.
- ActionEvent is the type of action.



Events

- Example of Events:
  - Clicking on a button
  - Pressing any key
  - Mouse movement
  - Scrolling
  - …
- Example on how to add a listener to a Button:

```
JButton fire = new JButton("fire");
fire.addActionListener(new FireButtonListener());
```

Inconsistent declarations

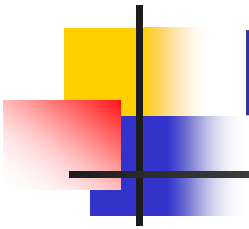
- The following is declared as an instance variable:
  - private JScrollBar slider = new JScrollBar(JScrollBar.VERTICAL, 45,5,0, 90);
- Whereas the following is declared as a local variable that belongs to a constructor:
  - JButton fire = new JButton("fire");
- They have been declared in different places, because local variables can be accessed only within the method in which they are declared in.

CannonWorld Source Code

Unit 2 – Section 3 - Key Terms

- Key Terms and Concepts to Remember:

cast	graphical user interface (GUI)	override
command-line argument	inner class	radians
coordinate system	interface	**super**
event	interface component	widget
event-driven interface	layout manager	wrapper class
event listener	listener objects	
event model	null	



Don’t Forget

- Do All the SAQs of Unit 2 and Unit 1 in case you missed them last week!!
- Experiment with JBuilder and try to comprehend how BallWorld and CannonGame execute and perform.
- Study all of Block 1 - Unit 2.