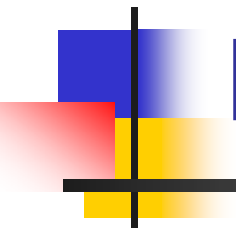
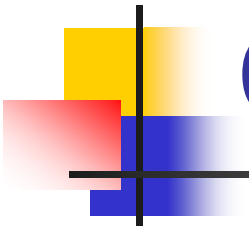


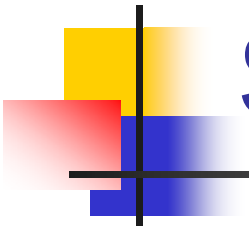
Block I – Unit 3

Using Inheritance



Concepts Covered

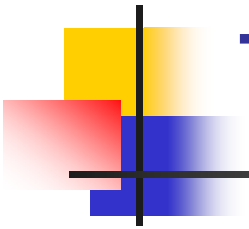
- Develop programs that will respond to mouse-related events.
- Use the Vector class for holding objects and primitive data types. (wrapper class)
- Write simple exception handlers.
- Develop programs using inheritance.
- Use polymorphism and polymorphic variables in your programs.



Study activity

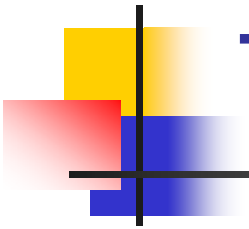
- Create PinBallGame1, PinBallGame2 and PinBallGame3 projects and run them.

(Exercises 5.1, 5.2 and 5.3 of the IDE Handbook)



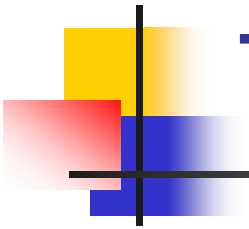
The game

- Users fire a ball from a small square.
- Balls encounters a variety of targets and fall back to the ground.
- User scores points when balls hit targets.
- The scores depends on the type of targets hit by the balls.



The Class PinBallFire

- Is a fire button for the application
- It knows how to draw itself.
- It can test a point to see if it is in the region of the fire button and returns a ball.



The PinBall Class

- Pinball extends the class CannonBall, even though there is no cannon in the pinball game.

- Fired from a PinBallFire and responded to gravity just like cannon ball.

The PinBall Class

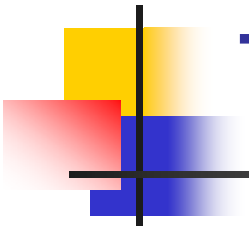
```
import java.awt.*;

public class PinBall extends CannonBall {
  public PinBall (Point loc) {
    super(loc, 8, -2 + Math.random(), -15);
  }

  public Rectangle box() {
    int r = radius();
    return new Rectangle(location().x-r, location().y-r, 2*r, 2*r);
  }
}
```

Method is inherited from the constructor in the parent class CannonBall

The CannonBall Class

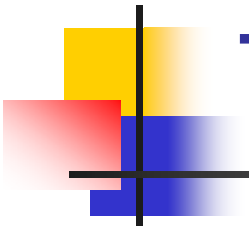
```
import java.awt.*;

public class CannonBall extends Ball {
   public CannonBall (Point loc, int r, double dx, double dy) {
      //invoke Ball's constructor
      super(loc, r);
      setMotion(dx, dy);
   }

   public double GravityEffect = 0.3;

   public void move () {
      changeInY += GravityEffect; //short for changeInY = changeInY +
     GravityEffect;
      super.move(); //update the ball position
   }
}
```

This is the code that gets executed from the previous call



The CannonBall Class

- super(loc, r);
- setMotion(dx, dy);
- There are two parts:
  - super invokes the constructor of its parent class Ball.
  - setMotion will set the direction of the initial move of the ball.

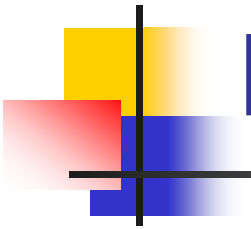


The Vector Class

- Need a structure that holds an un-determined number of balls.
- Unlike arrays, Vector can dynamically grow as elements are inserted.
- It could only hold objects or wrapped primitive data type.

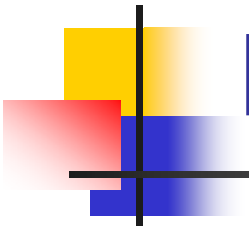
Accessing elements of Vector

- balls = new Vector();
- balls.addElement (newBall);
- For (int i = 0; i < balls.size(); i++)
- PinBall aBall = (PinBall) balls.elementAt(i);



Mouse Listeners

- The concept is that objects wait and listen.
- In our example, to create a listener, we define a class that implements the following interface.
- public interface MouseListener {
  public interface MouseListener {
  public void mousePressed(MouseEvent e);
  public void mouseReleased(MouseEvent e);
  public void mouseEntered(MouseEvent e);
  public void mouseExited(MouseEvent e);
  }



Mouse Listeners

- Instead of implementing all methods.
- Java provides a class named MouseAdapter.
- The class MouseAdapter implements MouseListener with empty methods.
- Programmer can inherit from MouseAdapter and implements the needed methods.

Mouse Listeners

- That is what we do in our example.
- We define an *inner* class that defines MouseListener by extending MouseAdapter.

```
Private class MouseKeeper extends MouseAdapter {
```

- An instance of this class is created and passed as argument to the method addMouseListener.

```
addMouseListener (new MouseKeeper());
```

Running the Application

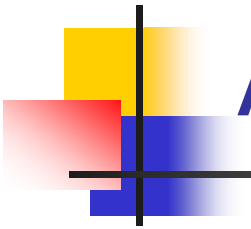
- The run method repeatedly calls moveBalls to move the balls, repaints the window and sleep shortly for window to be redrawn.

```
public void run() {
   while (true) {
     moveBalls();
     repaint();
     try {
       Thread.sleep(10);
     } catch(InterruptedException e) {System.exit(0);}
   }
 }
```

Running the Application

- The moveBalls routine cycles over the list of balls, moving each one.

```
private void moveBalls() {
   for (int i = 0; i < balls.size(); i++) {
     PinBall theBall = (PinBall) balls.elementAt(i);
     if (theBall.location().y < FrameHeight){
       theBall.move();
       for (int j = 0; j < targets.size(); j++) {
        PinBallTarget target =
           (PinBallTarget) targets.elementAt(j);
         if (target.intersects(theBall)) target.hitBy(theBall);
       }
     }
   }
}
```

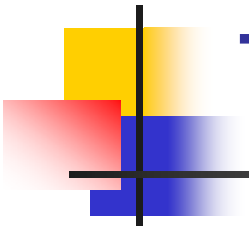


Adding targets

- As in real pinball game, we need to add targets for the ball to encounter.
- Some targets add scores, some move balls and some swallow the balls.
- We need a PinBallTarget interface.

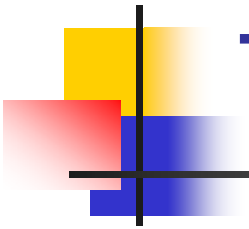
The PinBallTarget interface

- import java.awt.*;
- public interface PinBallTarget
- {
- public boolean intersects(PinBall aBall);
- public void moveTo(int x, int y);
- public void paint(Graphics g);
- public void hitBy(PinBall aBall);
- }
- The above defines a common behavior for Peg and Wall.



The Spring Class

- A Spring mimics the behaviour of a typical real-life pinball target which, when hit, gives the ball a slight ‘push’, that is, a small increase in velocity.



The Spring Class

- The Spring class implements PinBallTarget.
- It implements all methods.
- Describe all methods in the class.

Polymorphism / Polymorphic Variable

```
for (int j = 0; j < targets.size(); j++) {
     PinBallTarget target =
          (PinBallTarget) targets.elementAt(j);     //Line 1
        if (target.intersects(theBall)) target.hitBy(theBall);    //Line 2
}
```

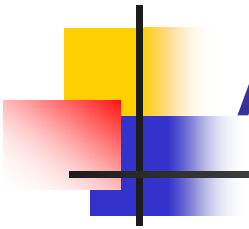
- The variable target is used to hold objects of type PinBallTarget .
- Line 1, extracts an individual target from the vector targets and calls it target which is of type PinBallTarget but actually holds targets of type Hole, ScorePad, Peg and so on.

Polymorphism / Polymorphic Variable

```
for (int j = 0; j < targets.size(); j++) {
      PinBallTarget target =
            (PinBallTarget) targets.elementAt(j);            //Line 1
          if (target.intersects(theBall)) target.hitBy(theBall);   //Line 2
}
```

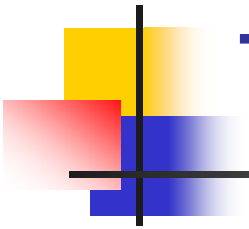
- Thus, since target can refer to objects of different types (though they must all be subtypes of PinBallTarget), target is said to be a polymorphic variable.

- In line 2, the fact that target is a polymorphic variable means that the methods intersects and hitBy actually used, are those that apply to whatever type of object target currently refers to.



An Aspect of Polymorphism.

- Spring and Wall both implements PinBallTarget.
- A variable declared as PinBallTarget could be holding either a Spring or a Wall.
- This is one aspect of polymorphism.



The Hole Class

public class Hole extends Ball implements PinBallTarget
{ .....

- Inheritance is used when we have structural relationship

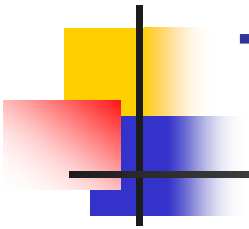
- Interface is used when we have behavioral relationship.

Inheritance and Interfaces

- The difference between implementing an interface and extending a class is subtle.
- Consider the definition of Hole:

```
public class Hole extends Ball implements PinBallTarget {
```

- The fact that Hole extends Ball means that, except where overridden, any method of Ball is available to objects of class Hole.
- However, the fact that Hole implements PinBallTarget means that Hole.java needs to implement the methods hitBy and intersects that are part of the PinBallTarget interface. If it does not, the class Hole would not compile. (moveTo and paint are already inherited from Ball)

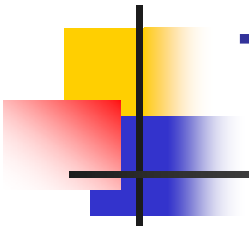


The ScorePad Class

- public class ScorePad extends Hole
- {.....

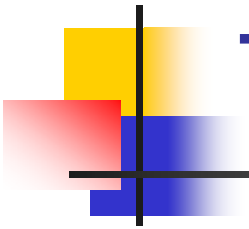
- Inherits the intersects behavior form Hole and moveTo from Ball

- But overrides the paint and hitBy methods that would otherwise be inherited by Hole



The Peg Class

- The Peg class is similar to ScorePad, therefore it extends ScorePad and overrides paint and hitBy.



The try statement

- Any code that can generate an exception should be placed in a try block.

- A try block can be followed by zero, one or more catch clauses. Each catch clause specifies the type of exception it can catch and an exception handler.

The try statement

- The optional finally block is executed when an exception that is not explicitly handled by one of the given catch clauses occurs.

```
try {
   sequence-of-statements;
} catch (SomeException ex1) {
      sequence-of-exception-handling-statements;
} catch (AnotherException ex2) {
      sequence-of-exception-handling-statements;
}
   .
   .
   finally {
      sequence-of-statements;
```

The variable world is static

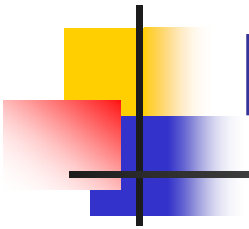
- The implementation of the method hitBy in ScorePad consists of the single message:

```
public void hitBy(PinBall aBall)
{
   PinBallGame.world.addScore(value);
}
```

- The method addScore is a member of the class PinBallGame

The variable world is static

- Since world is a public static variable defined in the class PinBallGame, it can be referred to in other classes by the construction: PinBallgame.world
- Any static variable (declared public) can be referred to in other classes by prefixing it with its class name.
- You have met a similar construction in
- System.out.println (where out is a static variable of the class System).



Members of ScorePad class

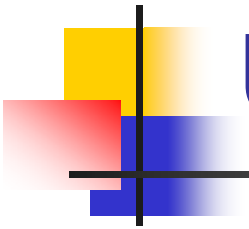
- The fact that the ScorePad class has members named intersects and moveTo is not immediately obvious from its class definition.

- These members are inherited from Hole and Ball respectively.

Message sends without an explicit receiver object

```
import java.awt.*;
public class Peg extends ScorePad
{
……..
public void hitBy (PinBall aBall)
    {
        super.hitBy(aBall);
        aBall.reflectVert();
        aBall.reflectHorz();
        while (intersects(aBall))
                aBall.move();
        state = 2;
    }
```

The message is sent to the current Peg object (this.intersects(aBall)



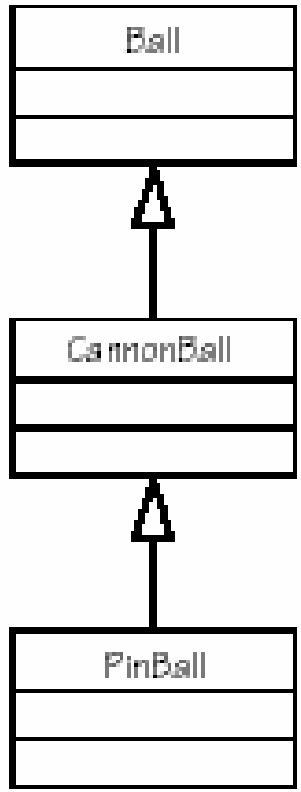
Understanding Inheritance

- Inheritance, we mean the property that instances of a child class can access both data and behavior associated with the parent class.

- Unless specified otherwise, all classes are derived from a single root class, named Object.

Understanding Inheritance

- Inheritance and Balls
- Consider the inheritance hierarchy
- All methods of Ball are available to objects of CannonBall.



Understanding Inheritance

- Describe sending the messages paint and move to object of CannonBall

```
public class Ball{
   public Ball (Point lc, int r) {loc = lc; rad = r;}
   public void setColor (Color newColor) {color = newColor;}
   public void setMotion (double dx, double dy) {
      changeInX = dx; changeInY = dy;
   }
   public void moveTo (int x, int y) {loc.move(x,y);}
   public void move (){loc.translate((int)changeInX, (int)changeInY);}
   public void paint (Graphics g) {    // method to display ball
      g.setColor (color);
      g.fillOval (loc.x-rad, loc.y-rad, 2*rad, 2*rad);
   }
}
```

```
public class CannonBall extends Ball {
   public CannonBall (Point loc, int r, double dx, double dy) {
      super (loc, r);            // invokes Ball's constructor
      setMotion (dx, dy);
   }
   public double GravityEffect = 0.3;
   public void move () {
      changeInY += GravityEffect;   // this statement is short for
                                    // changeInY = changeInY + GravityEf
      super.move ();                // update the ball position
   }
}
```

Understanding Inheritance

- Consider PinBall class: A new method, box , is defined to help detect whether a ball has encountered a target.

- If box was sent to object of CannonBall, an error will be reported.

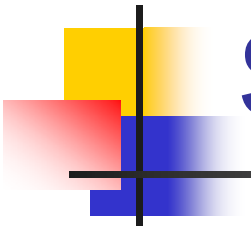
```
public class PinBall extends CannonBall {

    public PinBall (Point loc) {
        super (loc, 8, -2 + Math.random(), -15);
    }

    public Rectangle box () {
        int r = radius ();
        return new Rectangle (location ().x-r, location ().y-r, 2*r, 2*r);
    }
}
```

Different methods named intersects

```
public class Spring implements PinBallTarget
{
   private Rectangle pad;
.....

public boolean intersects (PinBall aBall)
{
   return pad.intersects(aBall.box());
}
```

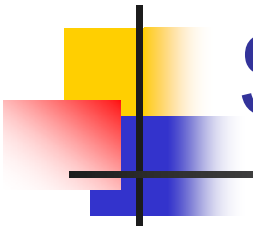
This is not the method being invoked by pad.intersects

Subtype, Subclass and Substitutability.



- A type of B is considered to be a subtype of A if:
  - An instance of B can be assigned to a variable declared as type A
  - That value can be used by the variable with no observable change in behavior.

Subtype, Subclass and Substitutability.



- A subclass refers to constructing a new class using inheritance (extends).

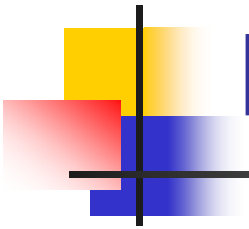
- In a majority of situations (not all), a subclass is also a subtype.

Subtype, Subclass and Substitutability.

- The idea is that the type given in a declaration of a variable does not have to match the type associated with the value the variable is holding.

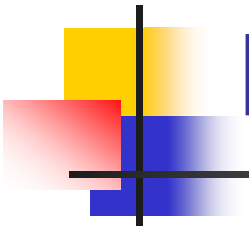
```
PinBallTarget target = (PinBallTarget) targets.elementAt(j);
```

Target holds a variety of different types of values. (Hole, Peg, Wall, ...).



Forms of Inheritance.

- Specialization
  - The child class is a subtype of the parent class. (extends)
- Specification
  - The parent defines behavior that is implemented in the child class. (implements)
- Construction
  - Child class makes use of the behavior of the parent class but is not a subtype of the parent class. (extends and implements. To inherit the methods from the interface that we do not want to implement)

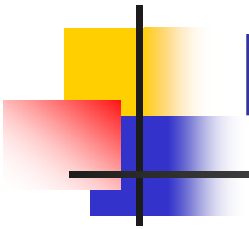


Forms of Inheritance.

- Extension
  - Child class adds new functionality to the parent class, but does not change any inherited behavior
- Limitation
  - The child class restricts the use of some of the behavior inherited form the parent class. (modify unwanted methods to give an error).
- Combination
  - Inheritance and interface. (extends and implements or implements more than one interface).

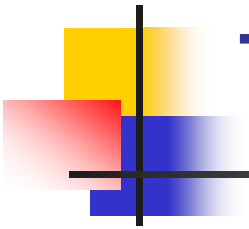
Modifiers and Inheritance.

- A reminder of a public feature (data field or method), a protected feature and a private feature.
- A static field is shared by all instances of a class.
- A static method can be invoked even when no instance has been created.
- Static data fields and methods are inherited the same way as nonstatic items, except that static methods cannot be overridden.



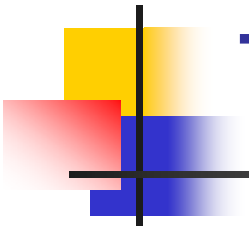
Modifiers and Inheritance.

- Both methods and classes can be declared to be abstract.
- An abstract class should not be instanciated.
- An abstract method must be overridden by a subclass.
- final is the opposite of abstract. When applied to a class, means the class cannot be subclassified.
- When applied to method, method cannot be overridden.



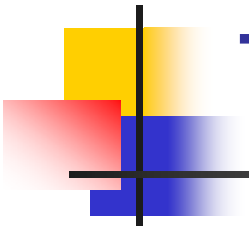
The benefit of Inheritance

- Software Reusability
- Increase Reliability
  - Code that execute frequently tends to have fewer bugs.
- Code Sharing
  - Two classes inherit from a single class.
- Consistency of Interface
  - Classes that inherit from the same class will in fact inherit the same behavior and therefore objects will have similar interfaces.



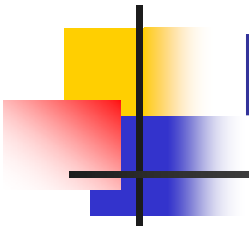
The benefit of Inheritance

- Software Components
  - Enables programmers to construct software using components.
- Rapid Prototyping
  - When software is constructed out of components, developers can concentrate on a new portion of the system.
- Polymorphism and Frameworks
  - Permits programmer to generate high-level reusable components that can be tailored to fit different applications by changing the low-level parts.
- Information Hiding
  - No need to understand the nature of the component that we reuse.



The cost of Inheritance

- Execution Speed
  - Inherited methods are often slower than specialized code.
- Program size
  - The use of software library imposes a size penalty.
- Message-Passing Overhead
  - Message passing is by nature a more costly operation
- Program Complexity



Lab Hours

- The extra sessions are now given in the form of practical activities in the Lab.

- Lab sessions are as follows:

Lab Hours

	Monday	Tuesday	Wednesday	Thursday	Friday
8:00 - 9:00					
9:00 - 10:00			M.K.		
10:00 - 11:00			M.K.		
11:00 - 12:00	A.H.				
12:00 - 1:00	A.H.			A.H.	
1:00 - 2:00				A.H.	
2:00 - 3:00		M.K.			
3:00 - 4:00		M.K.			
4:00 - 5:00	A.A.		A.A.		
5:00 - 6:00	A.A.		A.A.		
6:00 - 7:00					
7:00 - 8:00					
8:00 - 9:00					