Designing Classes

CSC 1051 – Data Structures and Algorithms I

Dr. Mary-Angela Papalaskari
Department of Computing Sciences
Villanova University

Course website:
www.csc.villanova.edu/~map/1051/
Where do objects come from?
Where do objects come from?

*Good question!*

*We will learn how to create a class that defines a new datatype, i.e., a new type of objects*

*We need to learn:*

1. What is the structure of a class definition?

2. How to specify what happens when an object is instantiated (i.e., when the `new` operator is used)?

3. How do we define the methods that can be invoked through objects of this class?
Example: **Account** datatype

- represents a generic bank account

```
acct1

acctNumber: 72354
balance: 102.56
name: "Ted Murphy"

acct2

acctNumber: 69713
balance: 40.00
name: "Jane Smith"
```
1. Structure of class definition

**Account class**

- **Data**
  - `int acctNumber;`
  - `double balance;`
  - `String name;`

- **Constructor**
- `deposit()`
- `withdraw()`
- `getBalance()`
- `toString()`

**Account object**

- `acctNumber`: 72354
- `balance`: 102.56
- `name`: "Ted Murphy"

- The object:
  - is like the house built from the blueprint
  - is an instance of the class
  - has its own data space & shares methods defined for this datatype

- The class is the **blueprint**

Classes define DATA and METHODS i.e., a **datatype**

CSC 1051 M.A. Papalaskari, Villanova University
2. Object instantiation

Creating Objects – OLD example:

- We have already seen something like this:

```java
Scanner scan = new Scanner (System.in);
```

This invokes the **Scanner constructor**, which is a special method that sets up the object.
A new Account object is created!

Account acct1 = new Account ("Ted Murphy", 72354, 102.56);

Creating Objects – our newly defined Account class:

Invokes the Account constructor, which is a special method that sets up the object

A new Account object is created!
As we have seen, once an object has been created, we can use the **dot operator** to invoke its methods:

```java
ans = scan.nextLine();

numChars = title.length();

amount = acct1.getBalance();

acct1.deposit(25.85);
```
Datatype / Client (also referred to as “servant / driver” classes)

Transactions.java
Author: MA Papalaskari
(based on Lewis/Loftus example)
// Demonstrates the creation and use of multiple Account objects.

public class Transactions {
  //-----------------------------------------------------------------
  // Creates some bank accounts and requests various services.
  //-----------------------------------------------------------------
  public static void main(String[] args) {
    Account acct1 = new Account("Ted Murphy", 72354, 102.56);
    Account acct2 = new Account("Jane Smith", 69713, 40.00);
    Account acct3 = new Account("Edward Demsey", 93757, 759.32);

    System.out.println(acct1);
    System.out.println(acct2);
    System.out.println(acct3);

    acct1.deposit(25.85);
    acct1.withdraw(60, 2.50);

    System.out.println();
    System.out.println(acct1);
    System.out.println(acct2);
    System.out.println(acct3);
  }
}

Datatype / Client

Account

Constructor

deposit()
withdraw()
getBalance()
toString()

import java.text.NumberFormat;
public class Account {
  int acctNumber;
  double balance;
  String name;

  public Account (String x, int y, double z) {
    name = x;
    acctNumber = y;
    balance = z;
  }

  public void deposit (double x) {
    balance = balance + x;
  }

  public void withdraw (double d, double c) {
    balance = balance - (d + c);
  }

  public String toString () {
    return name + "\n\nAccount Number: \n\nBalance: ";
  }
}

ACCOUNT

Account (also referred to as “servant / driver” classes)
import java.text.NumberFormat;

public class Account {
    int acctNumber;
    double balance;
    String name;

    public Account (String x, int y, double z) {
        name = x;
        acctNumber = y;
        balance = z;
    }

    public void deposit (double x) {
        balance = balance + x;
    }
}

// Constructor
// deposit()
// withdraw()
// getBalance()
// toString()
// Withdraws the specified amount from the account and applies the fee.
public void withdraw (double x, double fee) {
    balance = balance - x - fee;
}

// Returns the current balance of the account.
public double getBalance () {
    return balance;
}

// Returns a one-line description of the account as a string.
public String toString () {
    NumberFormat fmt = NumberFormat.getCurrencyInstance();
    return (acctNumber + "\t" + name + "\t" + fmt.format(balance));
}
public class Transactions {
    public static void main (String[] args) {
        Account acct1 = new Account ("Ted Murphy", 72354, 102.56);  
        Account acct2 = new Account ("Jane Smith", 69713, 40.00);  
        Account acct3 = new Account ("Edward Demsey", 93757, 759.32);  

        System.out.println (acct1);  
        System.out.println (acct2);  
        System.out.println (acct3);  

        acct1.deposit (25.85);  
        acct1.withdraw (60, 2.50);  

        System.out.println ();  
        System.out.println (acct1.toString());  
        System.out.println (acct2.toString());  
        System.out.println (acct3.toString());  
    }
}
public class Transactions {

    // Creates some bank accounts and requests various services.

    public static void main (String[] args) {
        Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
        Account acct2 = new Account ("Jane Smith", 69713, 40.00);
        Account acct3 = new Account ("Edward Demsey", 93757, 759.32);

        System.out.println (acct1);
        System.out.println (acct2);
        System.out.println (acct3);

        acct1.deposit (25.85);
        acct1.withdraw (60, 2.50);

        System.out.println () ;
        System.out.println (acct1.toString());
        System.out.println (acct2.toString());
        System.out.println (acct3.toString());
    }
}
Transactions class:

Creating Account objects

Account acct1 = new Account ("Ted Murphy", 72354, 102.56);

Account class

Constructor

public Account (String x, int y, double z)
{
    name = x;
    acctNumber = y;
    balance = z;
}

Constructor method

acct1

acctNumber 72354
balance 102.56
name "Ted Murphy"
Transactions class:

Creating more Account objects

Account acct1 = new Account ("Ted Murphy", 72354, 102.56);

Account acct2 = new Account ("Jane Smith", 69713, 40.00);
Account class: Using methods

```java
acct1.deposit (25.85);
//---------------------------------------------------
// Deposits the specified amount into the account.
//---------------------------------------------------
public void deposit (double x)
{
    balance = balance + x;
}
```
acct1.deposit (25.85);

// Deposits the specified amount into the account.
public void deposit (double x) {
    balance = balance + x;
}
acct1.withdraw (60, 2.50);

//------------------------------------------------
//  Withdraws the specified amount from the
//  account and applies the fee.
//------------------------------------------------
public void withdraw (double x, double fee)
{
    balance = balance - x - fee;
}
Account class: Another Example

acct1.withdraw (60, 2.50);

// Withdraws the specified amount from the account and applies the fee.
public void withdraw (double x, double fee) {
    balance = balance - x - fee;
}

acct1

acctNumber  72354
balance  65.91
name  "Ted Murphy"
double amount = acct1.getBalance();
// Note: this code is not used in Transactions.java

class Account {
    private double balance;
    public double getBalance () {
        return balance;
    }
}

class Transaction {
    Account acct1;
    public void addTransaction(amount) {
        acct1.balance += amount;
    }
}

CSC 1051 M.A. Papalaskari, Villanova University
The `toString()` method can be omitted! Nagual:

```java
public String toString ()
{
    NumberFormat fmt = NumberFormat.getCurrencyInstance();
    return (acctNumber +"\t"+ name +"\t"+ fmt.format(balance))
}
```

 acct1

 acctNumber 72354
 balance 102.56
 name "Ted Murphy"

System.out.println(acct1.toString());
Bank Account Example

• There are some improvements that can be made to the Account class

• The design of some methods could also be more robust, such as verifying that the amount parameter to the withdraw() method is positive

• Some of these improvements are in the book examples
  • Account.java, Transactions.java (simplified versions)
  • Account.java, Transactions.java (book versions)
public class RollingDice
{
    // Creates two Die objects and rolls them several times.
    public static void main (String[] args)
    {
        Die die1, die2;
        int sum;
        die1 = new Die();
        die2 = new Die();

        die1.roll();
        die2.roll();
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);

        die1.roll();
        die2.setFaceValue(4);
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);

        sum = die1.getFaceValue() + die2.getFaceValue();
        System.out.println ("Sum: " + sum);

        sum = die1.roll() + die2.roll();
        System.out.println ("Die One: " + die1 + ", Die Two: " + die2);
        System.out.println ("New sum: " + sum);
    }
}
public class Die {
    private final int MAX = 6;  // maximum face value

    private int faceValue;  // current value showing on the die

    public Die() {
        faceValue = 1;
    }

    public int roll() {
        faceValue = (int)(Math.random() * MAX) + 1;
        return faceValue;
    }
}

continue
continue

//------------------------------
// Face value mutator.
//------------------------------
public void setFaceValue (int value)
{
    faceValue = value;
}

//------------------------------
// Face value accessor.
//------------------------------
public int getFaceValue()
{
    return faceValue;
}

//------------------------------
// Returns a string representation of this die.
//------------------------------
public String toString()
{
    String result = Integer.toString(faceValue);
    return result;
}
NEXT: Focus on **Methods**

- Method control flow
- Method definition
  - Parameters
  - `return` statement
- Different ways of thinking about methods
Method Control Flow

- If the called method is in the same class, only the method name is needed.

```java
def myMethod():
    compute

myMethod();
```
Method Control Flow

• The called method is often part of another class or object

• Thus the dot operator is an addressing mechanism. Note that it can also be used to access an object’s or class’s data directly, for example
  – acct1.name
  – Color.black

• more on this later (encapsulation)
Invoking methods within the same class

- An object’s method may access any of the object’s other methods directly. Eg:

```java
public void addInterest(double rate) {
    deposit (rate*balance);
}
```

**Client code, eg: compound the interest for acct2 over 10 years**

```java
int year = 1;
while (year <= 10) {
    acct2.addInterest(0.03);
    year ++;
}
```
Method Control Flow: example

```java
acct2.addInterest(0.03);
```
**Method definition: Example**

- parameters
- return type
- return statement

```java
char ch = obj.calc (start, 2, "ABCDE");
```

```java
char calc (int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt (sum);

    return result;
}
```
More Method Examples:

- Write a method with two `double` parameters `a` and `b` that computes and returns the sum of squares of its two parameters (i.e., $a^2 + b^2$).

How do we invoke the method to compute & print: $(14.8)^2 + (37.65)^2$?
More Method Examples:

• Write a method with one \texttt{int} parameter \texttt{num}, that returns a \texttt{String} composed of “Happy Birthday” \texttt{num} times.

• How do we invoke the method to print “happy birthday” 4 times?
Getting to know classes so far

- Using predefined classes from the Java API.
- Defining classes for our own datatypes.

**datatypes:**
- Account
- Die
- Shoe
- Person

**Clients (Driver classes):**
- Transactions, OnePercent
- RollingDice
- YouVeGotShoes (Project)
- PeopleBeingPeople (Lab)

Next: UML class diagrams, visibility modifiers, graphical objects,
UML Class Diagrams

UML = Unified Modelling Language

- Example: A UML class diagram for the RollingDice program:

```java
public class RollingDice {
    public void main(String[] args) {
        Die faceValue = new Die()
        faceValue.ROLL();
        System.out.println(faceValue.toString());
    }
}
```

```
public class Die {
    private int faceValue;

    public void setFaceValue(int value) {
        this.faceValue = value;
    }

    public int getFaceValue() {
        return this.faceValue;
    }

    public String toString() {
        return this.faceValue;
    }
}
```
UML class diagram for Transactions program?
### Examples of datatypes (Classes)

<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Name</td>
<td>Set address</td>
</tr>
<tr>
<td></td>
<td>Address</td>
<td>Set major</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>Compute grade point average</td>
</tr>
<tr>
<td></td>
<td>Grade point average</td>
<td></td>
</tr>
<tr>
<td>Rectangle</td>
<td>Length</td>
<td>Set length</td>
</tr>
<tr>
<td></td>
<td>Width</td>
<td>Set width</td>
</tr>
<tr>
<td></td>
<td>Color</td>
<td>Set color</td>
</tr>
<tr>
<td>Aquarium</td>
<td>Material</td>
<td>Set material</td>
</tr>
<tr>
<td></td>
<td>Length</td>
<td>Set length</td>
</tr>
<tr>
<td></td>
<td>Width</td>
<td>Set width</td>
</tr>
<tr>
<td></td>
<td>Height</td>
<td>Set height</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute volume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute filled weight</td>
</tr>
<tr>
<td>Flight</td>
<td>Airline</td>
<td>Set airline</td>
</tr>
<tr>
<td></td>
<td>Flight number</td>
<td>Set flight number</td>
</tr>
<tr>
<td></td>
<td>Origin city</td>
<td>Determine status</td>
</tr>
<tr>
<td></td>
<td>Destination city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Current status</td>
<td></td>
</tr>
<tr>
<td>Employee</td>
<td>Name</td>
<td>Set department</td>
</tr>
<tr>
<td></td>
<td>Department</td>
<td>Set title</td>
</tr>
<tr>
<td></td>
<td>Title</td>
<td>Set salary</td>
</tr>
<tr>
<td></td>
<td>Salary</td>
<td>Compute wages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute bonus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute taxes</td>
</tr>
</tbody>
</table>
Encapsulation

• An encapsulated object can be thought of as a *black box* -- its inner workings are hidden from the client.

• The client invokes the interface methods which in turn manage the instance data.

**Violating Encapsulation**

It is possible for a class to access the instance data of another class directly.
Violating Encapsulation - **WRONG**

- It is possible for a class to access the instance data of another class directly – *but it’s not a good idea!*

```
acct1.name = "Joe";
```

---

```
Account.java
```
Use Accessors & Mutators - **RIGHT**

- Indirect access through methods
- accessors and mutators ("getters" and "setters")
- Usually named getX() and setX()

**Transactions.java**

```java
int x1 = acct1.getBalance();
```

**Account.java**

```java
deposit()
withdraw()
getBalance()
name
acctNumber
balance
```
Visibility Modifiers

• In Java, we enforce encapsulation through the appropriate use of *visibility modifiers*:
  
  – **public** – can be referenced from other classes
  
  – **private** – can be referenced only within that class:
  
  – **protected** – involves inheritance (discussed later)

• Data declared without a visibility modifier have *default visibility* and can be referenced by any class in the same package

• An overview of all Java modifiers is presented in Appendix E
Violating Encapsulation experiment

• Revisit the Account example
  (use our simplified versions from the course website or Lab 8)
• Add some code to the client (Transactions or Onepercent) to modify the value of an instance variable, eg:

```java
acct1.name = "Bernie";
```

• Run the program to verify that the name on that account has changed.
• Now modify Account.java – insert the modifier `private` in front of the instance variable declaration:

```java
private String name;
```

• Without changing anything in the client, re-compile the Account class and run your program again. Note the error you get:

Error: ____________________________________________________

CSC 1051 M.A. Papalaskari, Villanova University
Example: The Account class can have a constant for the interest rate:

```java
public final double RATE = 0.015;
```

A client (e.g., OnePercent.java) can access this constant directly:

```java
System.out.print ("Interest rate = " + acct1.RATE);
```
static Public Constants… BETTER

It is better to declare constants as static

```java
public final static double RATE = 0.015;
```

This way, a client can access the constants without creating an object, using the class name:

```java
System.out.println("Interest rate = " + Account.RATE);
```

CSC 1051 M.A. Papalaskari, Villanova University
## Visibility Modifiers – the RULES

<table>
<thead>
<tr>
<th></th>
<th>public</th>
<th>private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>NO (but OK for public constants)</td>
<td>Yes</td>
</tr>
<tr>
<td>Methods</td>
<td>Yes</td>
<td>Yes, for support methods only</td>
</tr>
</tbody>
</table>

See also [ImInUrClassMessingUrInstanceData.java](#)
Graphical Objects

• Some objects contain information that determines how the object should be represented visually

• Graphical objects
  – data about position, size, and other attributes
  – methods to draw the object

• Let's look at some other examples of graphical objects:

Example 1: SmilingFacePanel

  SmilingFace.java
  SmilingFacePanel.java

Example 2: SplatPanel

  Splat.java
  SplatPanel.java
  Circle.java
import javax.swing.JFrame;

public class SmilingFace
{
    //-----------------------------------------------------------------
    // Creates the main frame of the program.
    //-----------------------------------------------------------------
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Smiling Face");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        SmilingFacePanel panel = new SmilingFacePanel();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.JFrame;

public class SmilingFace {
    public static void main (String[] args) {
        JFrame frame = new JFrame("Smiling Face");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        SmilingFacePanel panel = new SmilingFacePanel();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
• The **SmilingFace** program draws a face by defining the `paintComponent` method of a panel.

• A UML class diagram:
import javax.swing.JPanel;
import java.awt.*;

public class SmilingFacePanel extends JPanel {
    private final int BASEX = 120, BASEY = 60; // base point for head

    public SmilingFacePanel () {
        setBackground (Color.blue);
        setPreferredSize (new Dimension (320, 200));
        setFont (new Font("Arial", Font.BOLD, 16));
    }
}

continue
// ------------------------------
//  Draws a face.
// ------------------------------

public void paintComponent (Graphics page) {
    super.paintComponent (page);

    page.setColor (Color.yellow);
    page.fillOval (BASEX, BASEY, 80, 80); // head
    page.fillOval (BASEX-5, BASEY+20, 90, 40); // ears

    page.setColor (Color.black);
    page.drawOval (BASEX+20, BASEY+30, 15, 7); // eyes
    page.drawOval (BASEX+45, BASEY+30, 15, 7);

    page.fillOval (BASEX+25, BASEY+31, 5, 5); // pupils
    page.fillOval (BASEX+50, BASEY+31, 5, 5);

    page.drawArc (BASEX+20, BASEY+25, 15, 7, 0, 180); // eyebrows
    page.drawArc (BASEX+45, BASEY+25, 15, 7, 0, 180);

    page.drawArc (BASEX+35, BASEY+40, 15, 10, 180, 180); // nose
    page.drawArc (BASEX+20, BASEY+50, 40, 15, 180, 180); // mouth
    page.setColor (Color.white);
    page.drawString ("Always remember that you are unique!",
                    BASEX-105, BASEY-15);
    page.drawString ("Just like everyone else.", BASEX-45, BASEY+105);
}
}
Jpanel Class – let’s look at the Java API

The **SmilingFacePanel** class is derived from the **JPanel** class using inheritance.

We are defining a subclass of **JPanel**.
Smiling Face Example

• Every Swing component has a `paintComponent` method

• The `paintComponent` method accepts a `Graphics` object that represents the graphics context for the panel

• We define the `paintComponent` method to draw the face with appropriate calls to the `Graphics` methods

• Note the difference between drawing on a panel and adding other GUI components to a panel
Objects with a draw() method

• The next example - Splat - is structured differently

• It draws a set of colored circles on a panel, but each circle is represented as a separate object that maintains its own graphical information

• The paintComponent method of the panel "asks" each circle to draw itself

• See Splat.java
• See SplatPanel.java
• See Circle.java
import javax.swing.*;
import java.awt.*;

public class Splat
{
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Splat");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new SplatPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.*;
import java.awt.*;

public class Splat {
    public static void main (String[] args) {
        JFrame frame = new JFrame ("Splat");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add (new SplatPanel());
        frame.pack();
        frame.setVisible (true);
    }
}
Splat

main (args : String[]) : void

SplatPanel

circle1: Circle
circle2: Circle
circle3: Circle
circle4: Circle
circle5: Circle

paintComponent(p: Graphics): void

Circle

diameter: int
color: Color
x: int
y: int

draw(p: Graphics): void
getDiameter(): int
getColor(): Color
getX(): int
getY(): int
setDiameter(size: int): void
setColor(shade: Color): void
setX(upperX: int): void
setY(upperY: int): void

CSC 1051 M.A. Papalaskari. Villanova
Splat

main (args : String[]) : void

SplatPanel

circle1: Circle
circle2: Circle
circle3: Circle
circle4: Circle
circle5: Circle

paintComponent(p: Graphics): void

Circle

diameter: int
color: Color
x: int
y: int
draw(p: Graphics): void
getDiameter(): int
getColor(): Color
getX(): int
getY(): int
setDiameter(size: int): void
setColor(shade: Color): void
setX(upperX: int): void
setY(upperY: int): void

circle1
circle2
circle3
circle4
circle5

diameter: 60
color: Red
x: 70
y: 35
diameter: 50
color: Green
x: 30
y: 20

circle2
circle3
circle4
circle5

diameter: 50
color: Blue
x: 20
y: 20

circle1
circle2
circle3
circle4
circle5

diameter: 60
color: Red
x: 70
y: 35
diameter: 50
color: Green
x: 30
y: 20

circle2
circle3
circle4
circle5

diameter: 50
color: Blue
x: 20
y: 20

circle1
circle2
circle3
circle4
circle5

diameter: 60
color: Red
x: 70
y: 35
diameter: 50
color: Green
x: 30
y: 20

circle2
circle3
circle4
circle5

diameter: 50
color: Blue
x: 20
y: 20

circle1
circle2
circle3
circle4
circle5

diameter: 60
color: Red
x: 70
y: 35
diameter: 50
color: Green
x: 30
y: 20

circle2
circle3
circle4
circle5

diameter: 50
color: Blue
x: 20
y: 20

CSC 1051 M.A. Papalaskari, Villanova
import javax.swing.*;
import java.awt.*;

public class SplatPanel extends JPanel {
    private Circle circle1, circle2, circle3, circle4, circle5;

    public SplatPanel() {
        circle1 = new Circle (30, Color.red, 70, 35);
        circle2 = new Circle (50, Color.green, 30, 20);
        circle3 = new Circle (100, Color.cyan, 60, 85);
        circle4 = new Circle (45, Color.yellow, 170, 30);
        circle5 = new Circle (60, Color.blue, 200, 60);

        setPreferredSize (new Dimension(300, 200));
        setBackground (Color.black);
    }

    continue
continue

//-----------------------------------------------------------------
//  Draws this panel by requesting that each circle draw itself.
//-----------------------------------------------------------------
public void paintComponent (Graphics page)
{
    super.paintComponent(page);

    circle1.draw(page);
    circle2.draw(page);
    circle3.draw(page);
    circle4.draw(page);
    circle5.draw(page);
}
}
import java.awt.*;

public class Circle
{
    private int diameter, x, y;
    private Color color;

    // Constructor: Sets up this circle with the specified values.
    public Circle (int size, Color shade, int upperX, int upperY)
    {
        diameter = size;
        color = shade;
        x = upperX;
        y = upperY;
    }

    continue
public void draw (Graphics page)
{
    page.setColor (color);
    page.fillOval (x, y, diameter, diameter);
}

public void setDiameter (int size)
{
    diameter = size;
}

public void setColor (Color shade)
{
    color = shade;
}
// X mutator.
public void setX (int upperX) {
    x = upperX;
}

// Y mutator.
public void setY (int upperY) {
    y = upperY;
}

// Diameter accessor.
public int getDiameter () {
    return diameter;
}

// Color accessor.
public Color getColor () {
    return color;
}

// X accessor.
public int getX () {
    return x;
}

// Y accessor.
public int getY () {
    return y;
}
continue

//-----------------------------------------------------------------------------
//  Color accessor.
//-----------------------------------------------------------------------------
public Color getColor ()
{
    return color;
}

//-----------------------------------------------------------------------------
//  X accessor.
//-----------------------------------------------------------------------------
public int getX ()
{
    return x;
}

//-----------------------------------------------------------------------------
//  Y accessor.
//-----------------------------------------------------------------------------
public int getY ()
{
    return y;
}