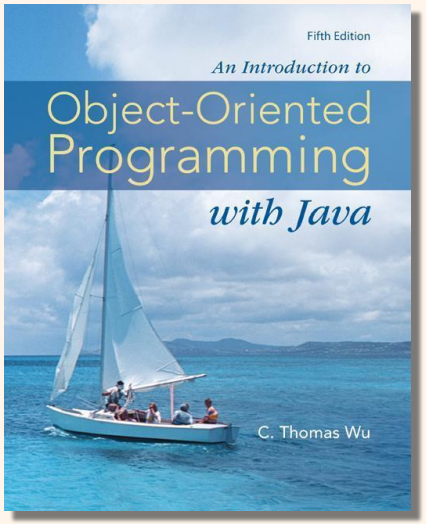


Chapter 5

Selection Statements

Objectives

After you have read and studied this chapter, you should be able to

- Implement a selection control using if statements
- Implement a selection control using switch statements
- Write boolean expressions using relational and boolean expressions
- Evaluate given boolean expressions correctly
- Nest an if statement inside another if statement
- Describe how objects are compared
- Choose the appropriate selection control statement for a given task
- Define and use enumerated constants

The if Statement

```
int testScore;

testScore = //get test score input

if (testScore < 70)

    System.out.println ("You did not pass" );

else

    System.out.println ("You did pass" );
```

This statement is executed if the testScore is less than 70.

This statement is executed if the testScore is 70 or higher.

Syntax for the if Statement

```
if ( <boolean expression> )

   <then block>
else

   <else block>
```

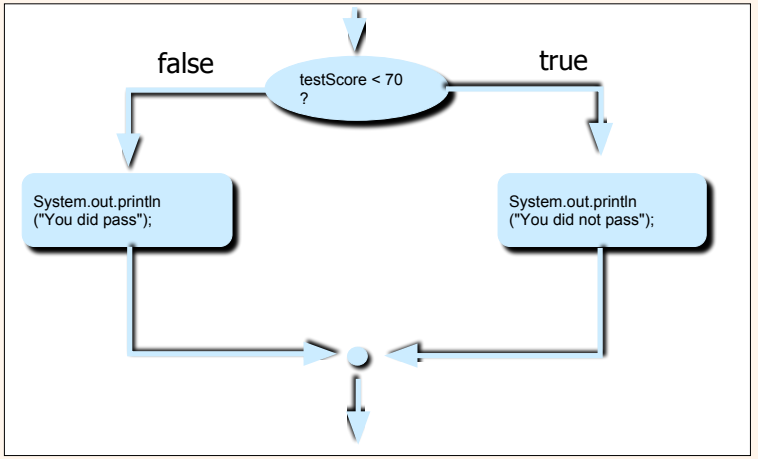
Boolean Expression

```
if (    testScore < 70    )
   System.out.println ("You did not pass");
else
   System.out.println ("You did pass");
```

Then Block

Else Block

Control Flow



Relational Operators

```
<   //less than
<=  //less than or equal to
==  //equal to
!=  //not equal to
>   //greater than
>=  //greater than or equal to
```

```
testScore < 80
testScore * 2 >= 350
30 < w / (h * h)
x + y != 2 * (a + b)
2 * Math.PI * radius <= 359.99
```


Compound Statements

- Use braces if the <then> or <else> block has multiple statements.

```
if (testScore < 70)
{
    System.out.println("You did not pass" );
    System.out.println("Try harder next time" );
}
else
{
    System.out.println("You did pass" );
    System.out.println("Keep up the good work" );
}
```

Then Block

Else Block

Style Guide

```
if ( <boolean expression> ) {
    …
} else {
    …
}
```

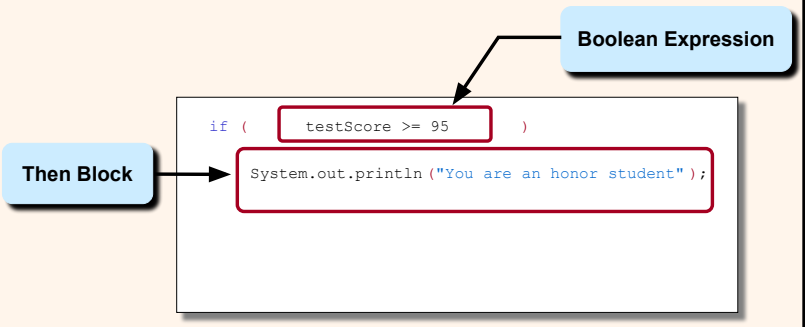
Style 1

```
if ( <boolean expression> )
{
    …
}
else
{
    …
}
```

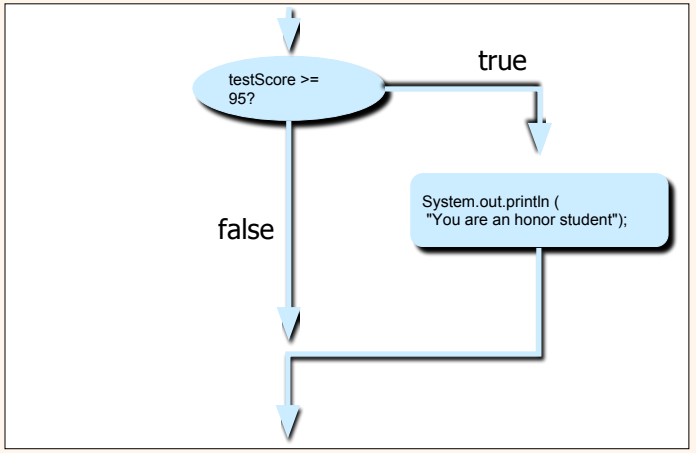
Style 2

The if-then Statement

```
if ( <boolean expression> )

    <then block>
```



Control Flow of if-then

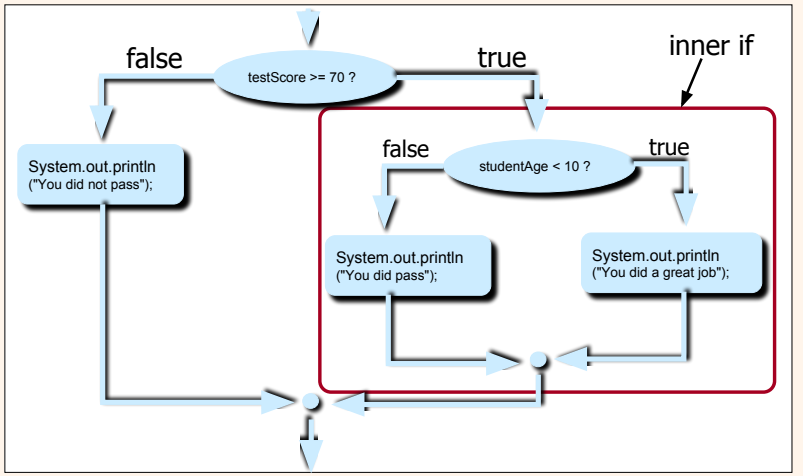


The Nested-if Statement

- The then and else block of an if statement can contain any valid statements, including other if statements. An if statement containing another if statement is called a nested-if statement.

```
if (testScore >= 70) {
     if (studentAge < 10) {
          System.out.println ("You did a great job");
     } else {
          System.out.println ("You did pass"); //test score >=
70
     }                          //and age >= 10
} else { //test score < 70
      System.out.println ("You did not pass");
}
```


Control Flow of Nested-if Statement



Writing a Proper if Control

```
if (num1 < 0)
     if (num2 < 0)
          if (num3 < 0)
               negativeCount = 3;
          else
               negativeCount = 2;
     else
          if (num3 < 0)
               negativeCount = 2;
          else
               negativeCount = 1;
else
     if (num2 < 0)
          if (num3 < 0)
               negativeCount = 2;
          else
               negativeCount = 1;
     else
          if (num3 < 0)
               negativeCount = 1;
          else
               negativeCount = 0;
```

```
negativeCount = 0;

if (num1 < 0)
        negativeCount++;
if (num2 < 0)
        negativeCount++;
if (num3 < 0)
        negativeCount++;
```

The statement

negativeCount++;

increments the variable by one

if – else if Control

Test Score	Grade
90 ≤ score	A
80 ≤ score • 90	B
70 ≤ score • 80	C
60 ≤ score • 70	D
score • 60	F

```
if (score >= 90)
     System.out.print("Your grade is A");

else if (score >= 80)
     System.out.print("Your grade is
B");

else if (score >= 70)
     System.out.print("Your grade is
C");

else if (score >= 60)
     System.out.print("Your grade is
D");

else
     System.out.print("Your grade is
F");
```


Matching else

Are (A) and (B) different?

(A)
```
if (x < y)
     if (x < z)
          System.out.print("Hello");
else
     System.out.print("Good bye");
```

(B)
```
if (x < y)
     if (x < z)
       System.out.print("Hello");
     else
       System.out.print("Good bye");
```

Both (A) and (B) means...

```
if (x < y) {
     if (x < z) {
        System.out.print("Hello");
     } else {
        System.out.print("Good
bye");
     }
}
```


Boolean Operators

- A *boolean operator* takes boolean values as its operands and returns a boolean value.
- The three boolean operators are
  - and: &&
  - or: ||
  - not !

```
if (temperature >= 22 && distanceToDestination < 2) {
    System.out.println("Let's walk");
} else {
    System.out.println("Let's drive");
}
```

Semantics of Boolean Operators

- Boolean operators and their meanings:

P	Q	P && Q	P || Q	!P
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

De Morgan's Law

- De Morgan's Law allows us to rewrite boolean expressions in different ways

```
Rule 1: !(P && Q) ⟷ !P || !Q

Rule 2: !(P || Q) ⟷ !P && !Q
```

```
!(temp >= 65 && dist < 2)

   ⟷!(temp >=22) || !(dist < 2) by Rule 1

   ⟷(temp < 22 || dist >= 2)
```


Short-Circuit Evaluation

- Consider the following boolean expression:

```
x > y || x > z
```

- The expression is evaluated left to right. If x > y is true, then there's no need to evaluate x > z because the whole expression will be true whether x > z is true or not.
- To stop the evaluation once the result of the whole expression is known is called *short-circuit evaluation*.
- What would happen if the short-circuit evaluation is not done for the following expression?

```
z == 0 || x / z > 20
```


Operator Precedence Rules

Group	Operator	Precedence	Associativity
Subexpression	( )	10 (If parentheses are nested, then innermost subexpression is evaluated first.)	Left to right
Postfix increment and decrement operators	++ --	9	Right to left
Unary operators	- !	8	Right to left
Multiplicative operators	* / %	7	Left to right
Additive operators	+ -	6	Left to right
Relational operators	< <= > >=	5	Left to right
Equality operators	== !=	4	Left to right
Boolean AND	&&	3	Left to right
Boolean OR	||	2	Left to right
Assignment	=	1	Right to left

Boolean Variables

- The result of a boolean expression is either **true** or **false**. These are the two values of data type **boolean**.
- We can declare a variable of data type **boolean** and assign a boolean value to it.

```
boolean pass, done;
pass = 70 < x;
done = true;
if (pass) {
    …
} else {
    …
}
```


Boolean Methods

- A method that returns a boolean value, such as

```
private boolean  isValid(int value) {
    if (value < MAX_ALLOWED)
        return true;
    } else {
        return false;
    }
}
```

Can be used as

```
if (isValid(30)) {
    …
} else {
    …
}
```


Comparing Objects

- With primitive data types, we have only one way to compare them, but with objects (reference data type), we have two ways to compare them.
  1. We can test whether two variables point to the same object (use ==), or
  2. We can test whether two distinct objects have the same contents.

Using == With Objects (Sample 1)

```
String str1 = new String("Java");
String str2 = new String("Java");

if (str1 == str2) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are not equal
```

Not equal because str1 and str2 point to different String objects.

Using == With Objects (Sample 2)

```
String str1 = new String("Java");
String str2 = str1;

if (str1 == str2) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are equal
```

It's equal here because str1 and str2 point to the same object.

Using equals with String

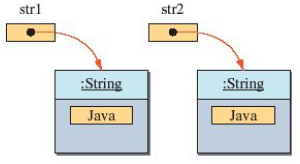
```
String str1 = new String("Java");
String str2 = new String("Java");

if (str1.equals(str2)) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are equal
```

It's equal here because str1 and str2 have the same sequence of characters.

The Semantics of ==

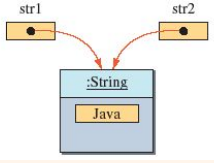
Case A: Two variables refer to two different objects.




```
String str1, str2;

str1 = new String("Java");
str2 = new String("Java");
```

str1 == str2 → false

Case B: Two variables refer to the same object.



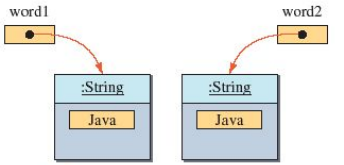

```
String str1, str2;

str1 = new String("Java");
str2 = str1;
```

str1 == str2 → true

In Creating String Objects

```
String word1, word2;

word1 = new String("Java");

word2 = new String("Java");
```

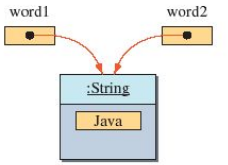
Whenever the new operator is used, there will be a new object.



word1 == word2 → false

```
String word1, word2;

word1 = "Java";

word2 = "Java";
```

Literal string constant such as "Java" will always refer to one object.



word1 == word2 → true

The switch Statement

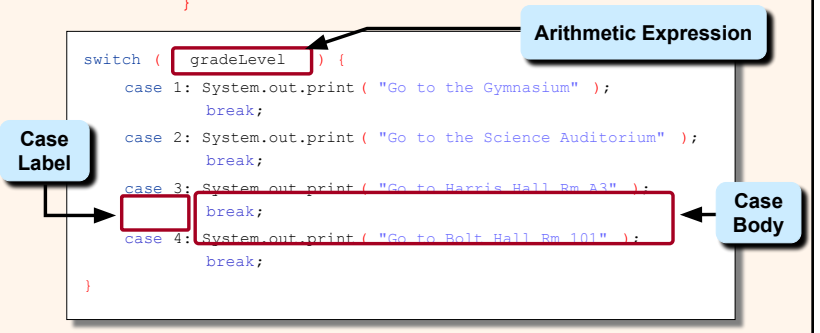
```
Scanner scanner = new Scanner(System.in);
System.out.println( "Ano (1,2,...):");
int gradeLevel = scanner.nextInt ();

switch (gradeLevel) {
    case 1: System.out.print ("Go to the Gymnasium");
            break;

    case 2: System.out.print ("Go to the Science Auditorium");
            break;

    case 3: System.out.print ("Go to Harris Hall Rm A3");
            Break;

    case 4: System.out.print ("Go to Bolt Hall Rm 101");
            break;
}
```

This statement is executed if the gradeLevel is equal to 1.

This statement is executed if the gradeLevel is equal to 4.

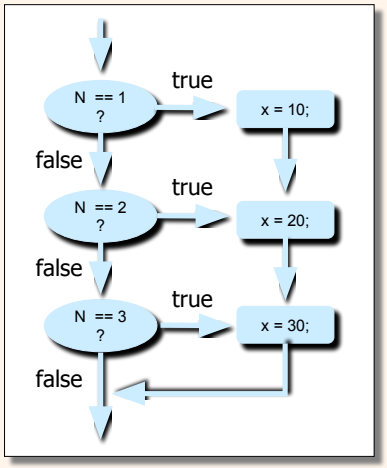
Syntax for the switch Statement

```
switch ( <arithmetic expression> ) {
    <case label 1> : <case body 1>
    …
    <case label n> : <case body n>
}
```



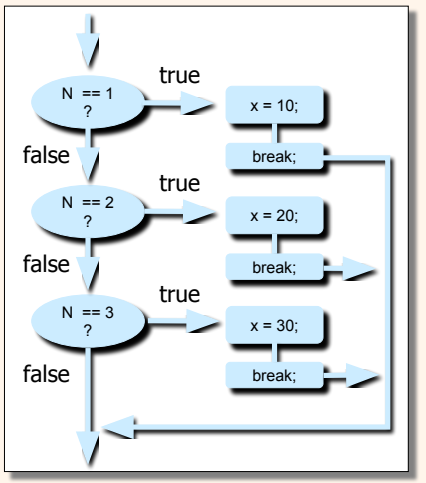
switch With No break Statements

```
switch ( N ) {
    case 1: x = 10;
    case 2: x = 20;
    case 3: x = 30;
}
```



switch With break Statements

```
switch ( N ) {
    case 1: x = 10;
            break;
    case 2: x = 20;
            break;
    case 3: x = 30;
            break;
}
```



switch With the default Block

```
switch (ranking) {
    case 10:
    case  9:
    case  8: System.out.print ("Master");
             break;

    case  7:
    case  6: System.out.print ("Journeyman");
             break;

    case  5:
    case  4: System.out.print ("Apprentice");
             break;

    default: System.out.print ("Input error: Invalid Data");
             break;
}
```


Drawing Graphics

- Chapter 5 introduces four standard classes related to drawing geometric shapes. They are
  - java.awt.Graphics
  - java.awt.Color
  - java.awt.Point
  - java.awt.Dimension
- These classes are used in the Sample Development section
- Please refer to Java API for details

Sample Drawing

```
import javax.swing.*; //for JFrame
import java.awt.*; //for Graphics and Container

class Ch5SampleGraphics {

    public static void main( String[] args ) {

        JFrame    win;
        Container contentPane;
        Graphics  g;

        win = new JFrame("My First Rectangle");
        win.setSize(300, 200);
        win.setLocation(100,100);
        win.setVisible(true);

        contentPane = win.getContentPane();
        g = contentPane.getGraphics();
        g.drawRect(50,50,100,30);
    }
}
```

win must be visible on the screen before you get its content pane.

The Effect of drawRect

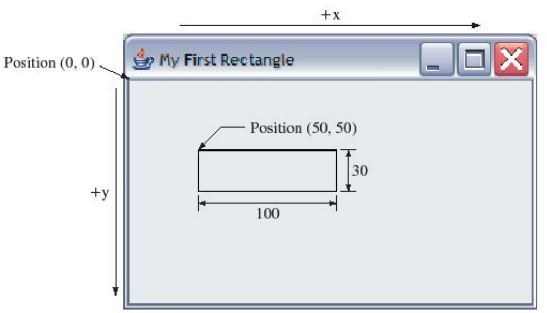
Syntax

A rectangle <width> wide and <height> high is displayed at position (<x>, <y>).

```
graphic.drawRect( <x>, <y>, <width>, <height>);
```

Example:

```
graphic.drawRect(50, 50, 100, 30);
```



Enumerated Constants

- In Chapter 3, we introduced numerical constants.
- Additional type of constants available in Java are called enumerated constants.
- Enumerated constants when used properly will support more reliable and robust programs.
- Enumerated constants are defined by using the reserved word enum.

Defining an Enumerated Type

- Consider the following example. Instead of defining numerical constants as

```
class Estacao {
    public static final int  INVERNO = 0;
    public static final int  PRIMAVERA = 1;
    public static final int  VERAO = 2;
    public static final int  OUTONO = 3;
}
```

- We can define an enumerated type as

```
class Estacao {
    public static enum EstacaoAno
                        {INVERNO, PRIMAVERA, VERAO, OUTONO }
}
```


Enumerated Types: More Examples

- Enumerated type is declared as

```
enum <enumerated type> { <constants> }
```

- Examples

```
enum Month {JANUARY, FEBRUARY , MARCH , APRIL, MAY, JUNE,
            JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER }

enum Gender {MALE, FEMALE}

enum SkillLevel {NOVICE, INTERMEDIATE , ADVANCED , EXPERT }
```


Using Enumerated Types

```
enum Fruit {APPLE, ORANGE , BANANA }
Fruit f1, f2, f3;
f1 = Fruit.APPLE;
f2 = f1;
System.out.println( "Favorite Fruit is " + f2);
Fruit favoriteFruit = …;

switch (favoriteFruit) {
   case Fruit.APPLE:  …
                      break;
   case Fruit.ORANGE: …
                      break;
   case Fruit.BANANA: …
                      break;
}
```

```
Favorite Fruit is APPLE
```

Accessing Enumerated Type from Outside

- If the enum type in a class is declared public, it can be accessed from outside the class

```
class Faculty {
    public static enum Rank {ASSISTENTE, AUXILIAR, ASSOCIADO, CATEDRATICO}
    . . .
}

class SampleMain {
    . . .
    Faculty.Rank  rank = Faculty.Rank.AUXILIAR;
    . . .
}
```

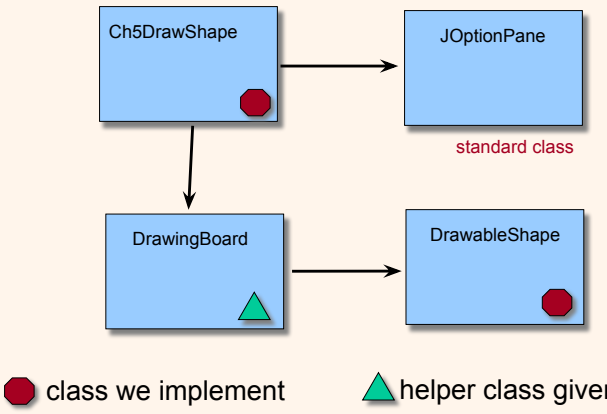

Problem Statement

Write an application that simulates a screensaver by drawing various geometric shapes in different colors. The user has an option of choosing a type (ellipse or rectangle), color, and movement (stationary, smooth, or random).

Overall Plan

- Tasks:
  - Get the shape the user wants to draw.
  - Get the color of the chosen shape.
  - Get the type of movement the user wants to use.
  - Start the drawing.

Required Classes



Development Steps

- We will develop this program in six steps:

1. Start with a program skeleton. Explore the DrawingBoard class.
2. Define an experimental DrawableShape class that draws a dummy shape.
3. Add code to allow the user to select a shape. Extend the DrawableShape and other classes as necessary.
4. Add code to allow the user to specify the color. Extend the DrawableShape and other classes as necessary.
5. Add code to allow the user to specify the motion type. Extend the DrawableShape and other classes as necessary.
6. Finalize the code by tying up loose ends.

Step 1 Design

- **The methods of the DrawingBoard class**
  - `public void addShape(DrawableShape shape)`
    Adds a shape to the DrawingBoard. No limit to the number shapes you can add
  - `public void setBackground(java.awt.Color color)`
    Sets the background color of a window to the designated color
  - `public void setDelayTime(double delay)`
    Sets the delay time between drawings to delay seconds
  - `public void setMovement(int type)`
    Sets the movement type to STATIONARY, RANDOM, or SMOOTH
  - `public void setVisible(boolean state)`
    Torna a janela visível, com particularidades
  - `public void start( )`
    Starts the drawing of added shapes using the designated movement type and delay time.

Step 1 Code

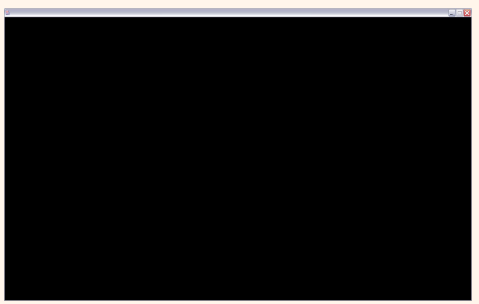
Program source file is too big to list here. From now on, we ask you to view the source files using your Java IDE.

Directory: Chapter5/Step1

Source Files: Ch5DrawShape.java

Step 1 Test

- In the testing phase, we run the program and verify that a DrawingBoard window with black background appears on the screen and fills the whole screen.



Step 2 Design

- **Define a preliminary DrawableShape class**
- The required methods of this class are
  - `public void draw(java.awt.Graphics g)`
    Draws a shape on Graphics object g.
  - `public java.awt.Point getCenterPoint( )`
    Returns the center point of this shape
  - `public java.awt.Dimension getDimension( )`
    Returns the bounding rectangle of this shape
  - `public void setCenterPoint(java.awt.Point pt)`
    Sets the center point of this shape to pt.

Step 2 Code

Directory: Chapter5/Step2

Source Files: Ch5DrawShape.java
DrawableShape.java

Step 2 Test

- We compile and run the program numerous times
- We confirm the movement types STATIONARY, RANDOM, and SMOOTH.
- We experiment with different delay times
- We try out different background colors

Step 3 Design

- We extend the main class to allow the **user to select a shape information**.
- We will give **three choices of shapes** to the user: Ellipse, Rectangle, and Rounded Rectangle
- We also need **input** routines for the user to enter the **dimension and center point**. The center point determines where the shape will appear on the DrawingBoard.
- Three input methods are

private int	inputShapeType( )
private Dimension	inputDimension( )
private Point	inputCenterPoint( )

Step 3 Code

Directory: Chapter5/Step3

Source Files: Ch5DrawShape.java
DrawableShape.java

Step 3 Test

- We run the program numerous times with different input values and check the results.
- Try both valid and invalid input values and confirm the response is appropriate

Step 4 Design

- We extend the main class to allow the **user to select a color**.
- We follow the input pattern of Step 3.
- We will allow the user to **select one of the five colors**.
- The color input method is

```
private Color      inputColor( )
```


Step 4 Code

Directory: Chapter5/Step4

Source Files: Ch5DrawShape.java
DrawableShape.java

Step 4 Test

- We run the program numerous times with different color input.
- Try both valid and invalid input values and confirm the response is appropriate

Step 5 Design

- We extend the main class to allow the user to **select a movement type**.
- We follow the input pattern of Step 3.
- We will allow the user to select one of the **three movement types**.
- The movement input method is

```
private int      inputMotionType( )
```


Step 5 Code

Directory: Chapter5/Step5

Source Files: Ch5DrawShape.java
DrawableShape.java

Step 5 Test

- We run the program numerous times with different movement input.
- Try both valid and invalid input values and confirm the response is appropriate

Step 6: Finalize

- Possible Extensions
  - Morphing the object shape
  - Changing the object color
  - Drawing multiple objects
  - Drawing scrolling text