

# Classes & objects

JC08-2.1

(A. Nguyen)

## Class vs. object

- A class represents a category of objects that share common **characteristics** and **behaviors**; it is a blue print for all **objects** in that **class**
- For example,
  - a class may be `BankAccount`
  - an **object** of that class may be `momsAccount` or `bobsAccount`

# The Java Library

- The **Java Library** is a collection of classes that are available for Java programmers to use
- Some examples:
  - Class `PrintStream` provides the printing capabilities (`println`, `print`), etc.
  - Class `JOptionPane` provides functionality with dialog windows, etc.
  - Class `String` provides functionality with handling pieces of text
- To see ***what*** is available and ***how*** to use, search “Java API” on the Internet
- **API** stands for **Application Programming Interface**

# The API

- Programmers consult the Java API when using the Java Library
- Programmers' own classes can be “automatically” documented in the same manner, including documentation within symbols `/** ... */`, called [Javadoc documentation](#) – you can easily switch between the source code and the documentation in BlueJ
- Thanks to the Javadoc documentation, past & present & concurrent programmers can “show” one another how to use their own classes

## The PrintStream class

- The `PrintStream` class provides the “printing” functionality to the console window, where the parameter/input may be of type `String` or `int`, etc.
- `System.out` is an object of the `PrintStream` class
- `PrintStream` has a method called `println` (& `print`, etc.), which can be called by any object of the `PrintStream` class
- To **invoke/call** the method, we write the object name, a period, the method name, and possibly the parameter(s) in parentheses; e.g.,  
`System.out.println("Hello");`

# The String class

- The `String` class represents a sequence of text, with various methods/functionalities relating to manipulating the text
- Some functionalities are: finding certain (smaller) sequence of text, extracting (smaller) sequence of text, etc.
- When a programmer writes a piece of text in double quotes in a Java program, a `String` **object** is automatically created to represent that text

## The Scanner class

- The class `PrintStream` provides the functionality to display the output to the console window; the `Scanner` class provides the functionality to accept input from the user (from the keyboard, etc.)
- In preparing the keyboard for input, a programmer creates a **keyboard object**:  

```
Scanner kboard = new Scanner(System.in);
```
- When done, the programmer writes:  

```
kboard.close();
```
- In between these two statements, the programmer can accept inputs from the user, of various data types (text, integer, etc.)

## The Scanner class (cont.)

- After the keyboard object, called `kboard`, is created, the programmers use the following statements to accept inputs from the user:

```
String name = kboard.nextLine(); // Get text & save in name  
int age = kboard.nextInt(); // Get integer & save in age  
double dist = kboard.nextDouble(); // Get floating-point # & save in dist
```

- Note that, before accepting the user's input, you must remember to ask/request; e.g., `System.out.print("Your name? ");`
- Certain library classes (e.g., `String`) are automatically included in a Java program. The `Scanner` class is not. Therefore we must import it, before the class header: `import java.util.Scanner;`



# Variables

JC08-2.2

(A. Nguyen)

# Data types

- There are 2 kinds of data types: primitive data types and classes
- There are 8 **primitive data types** in Java :  
    `int, double, boolean, char,`  
    `byte, short, long, float`
- Some **classes** from the Java Library are `String, Scanner,`  
    `Color, Rectangle`

## Data types (cont.)

- In the following statements:

```
String name = kboard.nextLine(); // Get text & save in name
```

```
int age = kboard.nextInt(); // Get integer & save in age
```

```
double dist = kboard.nextDouble(); // Get floating-point # & save in dist
```

... name is an **object** (aka **instance of a class**) of the `String` class

... age is a **variable of primitive data type** called `int`

... dist is a **variable of primitive data type** called `double`

## Data types (cont.)

- Classes come from the Java Library (e.g., `String`, `Scanner`, etc.), and also from creation by programmers/you (e.g., `HelloPrinter`, etc.)
- Different data types need different amounts of RAM space
- Note that `String` is a class; `char` is a primitive data type:  

```
String firstName = "Abraham"; // in double quotes  
char lastInitial = 'L';        // in single quotes
```

# Variable names

- A variable is a piece of RAM space where we store some data.
- A variable is named, based on the java syntax & convention:
  - May contain letters and numbers and underscore (\_); no space or special characters – Names are case-sensitive: variable `area` is NOT variable `Area`
  - Must start with a letter, in camel case
  - Should be meaningful, and cannot be a reserved word (like `int`, `class` etc.)
- Good variable names: `firstName`, `score1`, `score2`
- Bad variable names:
  - `6pack` – **syntax error**: name cannot start with a number
  - `stuff` – bad style/convention: name is **not meaningful**
  - `car color` – **error**: space not allowed; use `carColor`

## Declaration: asking for RAM space

- A **declaration** is a **request for some RAM space**
- Declaration is composed of **at least** the following:
  - `<dataType> <variableName>;` // for a local variable
  - `<accessSpecifier> <dataType> <variableName>;` // for a field
- Example:
  - `int numOfYrs; // request RAM space for primitive data type int`
  - `String firstName; // request RAM space for an address (or reference) of String class`
- A variable must be declared ***before*** being used
- A **local variable** (i.e., of the same name) cannot be declared more than once – or else **syntax error**

## Declaration: logic error

- When a “field” is “redeclared” inside, say, a constructor (syntactically correct), it is NOT a field here; it is a local variable
- So when this local variable is assigned to some value, the field (with the same name) is NOT

# Assignment

- After a variable is **declared**, we can **assign** a value to it (using the equal sign):  
`numOfYrs = 10;`
- Note that this equal sign does NOT mean “equal” as in math; it means “**is assigned to**” or “**gets/has the value of**”
- Later in the code, the variable may be assigned to another value (**do NOT declare it again**):

```
numOfYrs = numOfYrs * 2; // double the # of years
```

- A declaration and an assignment may be combined into 1 statement:  
`int numOfYrs = 10;`



## Assignment (cont.)

- In an assignment statement, the left side of the symbol ***must*** be a variable name; the right side may be one of the following:

```
numOfYrs = 10;           // constant
```

```
sumOfInts = 1 + 2 + 3;   // arithmetic expression
```

```
numOfYrs = initValue;    // another variable, previously declared & assigned
```

```
age = kboard.nextInt();  // result from a method call
```

- Bad assignment statements:

```
numOfYrs = "Hello";      // data type of value is not same as data type of variable
```

```
1 + 2 + 3 = sumOfInts;    // the variable must be on the left
```

# Method call

JC08-2.3

(A. Nguyen)

# Method signatures

- A class has 2 main parts:
  - **Instance variables**, aka **fields**, representing characteristics
  - **Methods**, representing behaviors
- The API for a class shows both
- Each method is documented with the method name, parameters/inputs (inside parentheses), and return value/output

# Some String methods

- From the Java API:

String	<code>toUpperCase()</code> Converts all of the characters in this String to upper case using the rules of the default locale.
String	<code>toLowerCase()</code> Converts all of the characters in this String to lower case using the rules of the default locale.
String	<code>substring(int beginIndex)</code> Returns a new string that is a substring of this string.
String	<code>substring(int beginIndex, int endIndex)</code> Returns a new string that is a substring of this string.
int	<code>length()</code> Returns the length of this string.
String	<code>replace(char oldChar, char newChar)</code> Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

## Examples of method signatures

- Example: a method (**replace**) may accept **parameters/arguments/inputs**. (**oldChar** & **newChar** of type **char**), and returns a value of type **String**, as shown in the API:

```
String      replace(char oldChar, char newChar)  
           Returns a new string resulting from replacing all occurrences of oldChar  
           in this string with newChar.
```

- Example: `Scanner` has one method called `nextInt`; it takes no parameters (empty parentheses) and returns a value of type `int`, as shown in the API:

```
int          nextInt()  
           Scans the next token of the input as an int.
```

# Calling methods

- To call a method (of a class) is to ask the object of that class to carry out the instructions in that method
- For example, after creating a Scanner object:  
`Scanner kboard = new Scanner(System.in);`  
... we can ask `kboard` to get the input by making an appropriate method call:  
`String name = kboard.nextLine(); // get text`  
`int age = kboard.nextInt(); // get integer`  
etc.
- **Note the syntax:** object name, dot, method name, parameters in parentheses (if any)

## Calling methods (cont.)

- The call to that method must provide appropriate parameters:

```
int nameLen = name.length(); // no param to String name
System.out.print("Hello");    // 1 param
```

- A method may **return** a value (i.e., one output) or not. In the method signature, if it returns a value, the data type is shown; if not, `void` is shown

```
int nameLen = name.length(); // with return value
System.out.print("Hello");    // without return value
```

# Constructing objects

JC08-2.4

(A. Nguyen)



# Constructor

- A Java program is a “world” of objects that interact with one another
- Before an object can exist (to interact), it **must be created**, which is done with the **new** operator and a call to the class’s constructor

- Example:

`Rectangle rect = new Rectangle(5, 10, 20, 30);`

class

object

constructor

“new” operator

Parameters/arguments/inputs

- A constructor is a special “method” whose purpose is to initialize the **instance variables** (aka **fields**) of a newly created object
- Just about all objects must be created this way – except the String class: `String school = “Cal”;` // automatically did “new”

## Constructor (cont.)

- A constructor's name is the same as the name of the class and the .java file; class `Rectangle` is in file `Rectangle.java` and has constructor `Rectangle`
- Like a method, a constructor may accept parameter(s) or not
- The result of `new` and a constructor call is a newly created object:

```
new Rectangle(5, 10, 20, 30);
```

- ... and you can store it in a variable, like `rect`:

```
Rectangle rect = new Rectangle(5, 10, 20, 30);
```

# Constructors of class Rectangle

## Constructors

### Constructor and Description

`Rectangle()`

Constructs a new Rectangle whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are both zero.

`Rectangle(Dimension d)`

Constructs a new Rectangle whose top left corner is (0, 0) and whose width and height are specified by the Dimension argument.

`Rectangle(int width, int height)`

Constructs a new Rectangle whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are specified by the arguments of the same name.

`Rectangle(int x, int y, int width, int height)`

Constructs a new Rectangle whose upper-left corner is specified as (x,y) and whose width and height are specified by the arguments of the same name.

`Rectangle(Point p)`

Constructs a new Rectangle whose upper-left corner is the specified Point, and whose width and height are both zero.

`Rectangle(Point p, Dimension d)`

Constructs a new Rectangle whose upper-left corner is specified by the Point argument, and whose width and height are specified by the Dimension argument.

`Rectangle(Rectangle r)`

Constructs a new Rectangle, initialized to match the values of the specified Rectangle.

## Syntax 2.3 Object Construction

Syntax `new ClassName(arguments)`

The new expression yields an object.

Construction arguments

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

Usually, you save the constructed object in a variable.

```
System.out.println(new Rectangle());
```

You can also pass a constructed object to a method.

Supply the parentheses even when there are no arguments.

# Accessors & Mutators

JC08-2.5

(A. Nguyen)

# The Rectangle API

## Methods

Modifier and Type	Method and Description
void	<code>add(int newx, int newy)</code> Adds a point, specified by the integer arguments <code>newx</code> , <code>newy</code> to the bounds of this Rectangle.
void	<code>add(Point pt)</code> Adds the specified Point to the bounds of this Rectangle.
void	<code>add(Rectangle r)</code> Adds a Rectangle to this Rectangle.
boolean	<code>contains(int x, int y)</code> Checks whether or not this Rectangle contains the point at the specified location (x,y).
boolean	<code>contains(int X, int Y, int W, int H)</code> Checks whether this Rectangle entirely contains the Rectangle at the specified location (X,Y) with the specified dimensions (W,H).
boolean	<code>contains(Point p)</code> Checks whether or not this Rectangle contains the specified Point.
boolean	<code>contains(Rectangle r)</code> Checks whether or not this Rectangle entirely contains the specified Rectangle.
Rectangle2D	<code>createIntersection(Rectangle2D r)</code> Returns a new Rectangle2D object representing the intersection of this Rectangle2D with the specified Rectangle2D.
Rectangle2D	<code>createUnion(Rectangle2D r)</code> Returns a new Rectangle2D object representing the union of this Rectangle2D with the specified Rectangle2D.

## The Rectangle API (cont.)

boolean	<code>equals(Object obj)</code> Checks whether two rectangles are equal.
Rectangle	<code>getBounds()</code> Gets the bounding Rectangle of this Rectangle.
Rectangle2D	<code>getBounds2D()</code> Returns a high precision and more accurate bounding box of the Shape than the <code>getBounds</code> method.
double	<code>getHeight()</code> Returns the height of the bounding Rectangle in double precision.
Point	<code>getLocation()</code> Returns the location of this Rectangle.
Dimension	<code>getSize()</code> Gets the size of this Rectangle, represented by the returned Dimension.
double	<code>getWidth()</code> Returns the width of the bounding Rectangle in double precision.
double	<code>getX()</code> Returns the X coordinate of the bounding Rectangle in double precision.
double	<code>getY()</code> Returns the Y coordinate of the bounding Rectangle in double precision.
void	<code>grow(int h, int v)</code> Resizes the Rectangle both horizontally and vertically.
boolean	<code>inside(int X, int Y)</code> Returns true if the point (X, Y) is inside the Rectangle.

## The Rectangle API (cont.)

void	<code>setBounds(int x, int y, int width, int height)</code> Sets the bounding Rectangle of this Rectangle to the specified x, y, width, and height.
void	<code>setBounds(Rectangle r)</code> Sets the bounding Rectangle of this Rectangle to match the specified Rectangle.
void	<code>setLocation(int x, int y)</code> Moves this Rectangle to the specified location.
void	<code>setLocation(Point p)</code> Moves this Rectangle to the specified location.
void	<code>setRect(double x, double y, double width, double height)</code> Sets the bounds of this Rectangle to the integer bounds which encompass the specified x, y, width, and height.
void	<code>setSize(Dimension d)</code> Sets the size of this Rectangle to match the specified Dimension.
void	<code>setSize(int width, int height)</code> Sets the size of this Rectangle to the specified width and height.
void	<code>translate(int dx, int dy)</code> Translates this Rectangle the indicated distance, to the right along the X coordinate axis, and downward along the Y coordinate axis.
Rectangle	<code>union(Rectangle r)</code> Computes the union of this Rectangle with the specified Rectangle.



# Accessors

- A method is an **accessor** if it **does not change** the value of the field(s)
- In the `Rectangle` class, some accessors are: `getHeight`, `getWidth`, `getX`, `getY`
- An accessor's name usually starts with `get`
- All `String` methods are accessors. I.e., when a method is called, the original text is unchanged; the method returns new text
- The `String` class is called **immutable**, i.e., the fields are never changed

# Mutators

- A method is an **mutator** if it **changes** the value of the field(s)
- In the Rectangle class, some mutators are: `setHeight`, `setWidth`, `setX`, `setY`, `setBounds`, `setLocations`, `add`, `grow`, `translate`, `union`
- Some accessors names start with `set`

# Testing

JC08-2.7

(A. Nguyen)

# Tester

- A Tester is a (main) program that is used to test *all the methods* of a class
- When a method returns a value, print it to the console window, along with the “expected” value that you/programmer have calculated to verify
- When printing something, remember to start with a label to identify the meaning of the value

# Test Script

- Programs are usually more complicated than displaying one result; so we use a test script
- A **test script** is a list of inputs and expected outputs, of different situations:
  - General cases
  - Boundary cases
- When testing, the programmer gives the program input(s), then compares the output(s) from the program with the output(s) from the test script to verify

# Object Reference

JC08-2.8

(A. Nguyen)

# Primitive data type

- When a variable is of primitive data type, its value is stored directly in the memory slot

- Example: `int width;`

`width = 10;`

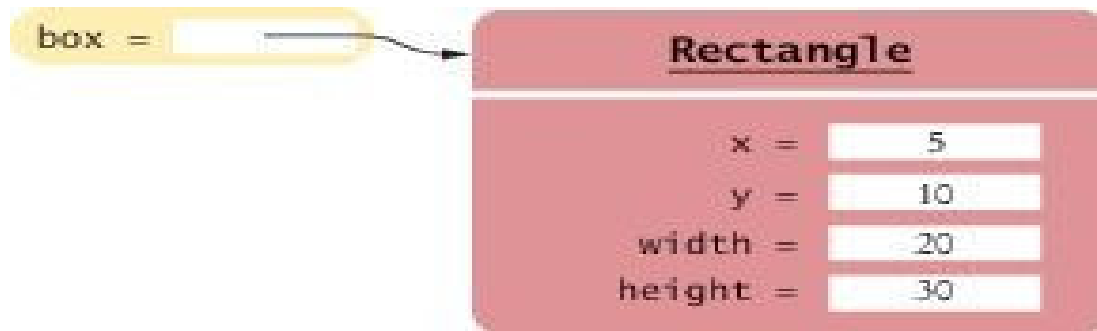


`width = 20;`



# Object

- When a variable is an object, its value is stored via the [address/reference](#) to the memory slot
- Example: `box` is an object of the `Rectangle` class, with top left corner at (5, 10), width 20, length 30:



- See Fig. 19 on p. 57 – **Be careful**



Graphics class & others

## The Color class

- Must import `java.awt.Color;`
- Search for “java api” on the Internet, and click on the `Color` class to see the documentation for it
- Some default colors are `Color.BLACK`, `Color.ORANGE`
- There are 13 default colors: `Color.BLACK ... Color.YELLOW`
- Java is case-sensitive, but both `Color.BLACK` and `Color.black` are present in the API, so they both refer to the black color. However, it is better to use the upper-case name, which follows the naming convention for “constants”

## The Color class (cont.)

- An object of the Color class is composed of 3 pieces of info: red, green, blue, each of which has value 0-255, referred to as RGB

- To “create” color from scratch, call

`Color(int r, int g, int b)` with the new operator:

```
Color color1 = new Color(255, 0, 0); // is Color.RED
Color color2 = new Color(200, 150, 180); //purplish
Color color3 = new Color(255, 255, 255); // is Color.WHITE
```

## The Color class (cont.)

- To “create” an opaque color from scratch, call constructor `Color(int r, int g, int b, int a)` with the `new` operator, where 4<sup>th</sup> parameter `a`, with value 0 – 255, determines how opaque it is:

```
Color color4 = new Color(255, 255, 255, 200);
```

## The Rectangle class

- Must import `java.awt.Rectangle;`
- Create a rectangle by calling  
`Rectangle(int x, int y, int width, int height),`  
where `x` and `y` are top-left corner:  
`Rectangle box1 = new Rectangle(5, 10, 200, 300);`
- There are also other `Rectangle` constructors – see Java API
- Some common methods are: `grow`, `setSize`, `setLocation`, `getX`, `getY`, `getWidth`, `getHeight`, `translate`
- If you have 2 shapes of the same size but at different locations (like 2 windows in a house), create one `Rectangle` object, draw it, translate it, and draw it again

## The Ellipse.Double class

- Must import `java.awt.geom.Ellipse2D;`
- Create an ellipse by calling `Ellipse2D.Double(double x, double y, double w, double h)`, where `x` & `y` are top-left corner and `w` & `h` are width & height:  

```
Ellipse2D.Double oval1 = new Ellipse2D.Double(150, 250, 100, 50);
```
- There is also another `Ellipse2D.Double` constructor – see Java API
- Some common methods are similar to those of `Rectangle`

## The Line.Double class (cont.)

- **Must import** `java.awt.geom.Line2D.Double;`
- **Create a line by calling**  
`Line2D.Double(double x1, double y1, double x2, double y2)`, where the parameters are the coordinates of the 2 endpoints:

```
Line2D.Double aLine = new Line2D.Double(50, 100, 95, 75);
```

# The Polygon class

- Must import `java.awt.Polygon;`
- Create a line by calling `Polygon(int[] xpoints, int[] ypoints, int npoints)`, where the parameters are x-coordinates, y-coordinates, number of points:

```
int[] xCoords = {120, 140, 130};
int[] yCoords = {150, 190, 220};
Polygon aPoly = new Polygon(xCoords, yCoords, 3);
```
- Some common methods are: `translate`, `addPoint` – see Java API



## The Graphics2D class

- Drawing is done in a subclass of JComponent

- Must

```
import java.awt.Graphics;  
import java.awt.Graphics2D;  
import java. ; // for JComponent
```

- Cast:

```
Graphics2D g2 = (Graphics2D) g;
```

## The Graphics2D class (cont.)

- We will use `g2` as the variable name for the object of `Graphics2D` in these examples
- To draw a shape object named `box`, call `g2.draw(box)` ;
- To fill a shape object named `oval`, call `g2.fill(box)` ;
- To draw text, call `g2.drawString("9870", 50, 130)` ;
- What is drawn/filled before will be under those drawn/filled after; e.g., if 2 concentric circles are filled and if the smaller circle is filled before the larger circle, then only the larger circle is visible

## The Graphics2D class (cont.)

- To draw or fill in certain color named `aColor`, set the color BEFORE drawing or filling, by calling `g2.setColor(aColor);`
- To draw with thickness 5, set the thickness BEFORE drawing by calling `g2.setStroke(new BasicStroke(5));`
- To use `BasicStroke`, must  
`import java.awt.BasicStroke;`
- Note that the color & thickness are used until the next setting

Using the Random class

# The Random class

- **Must import `java.util.Random`;**
- **To create an object:**  
`Random randGen = new Random(); // create object`
- **To request an int between 0 and n, including 0 but not n, (where `int n=6`):**  
`int result = randGen.nextInt(n); // get 0 to 5, inclusively`
- **Adding 1 to result will simulate the roll of a 6-sided die:**  
`int result = randGen.nextInt(n) + 1; // get 1 to 6, inclusively`
- **Consult the API to get a random number of different types**

THE END