

BBM 102 – Introduction to Programming II

Classes and Objects in Java



Today

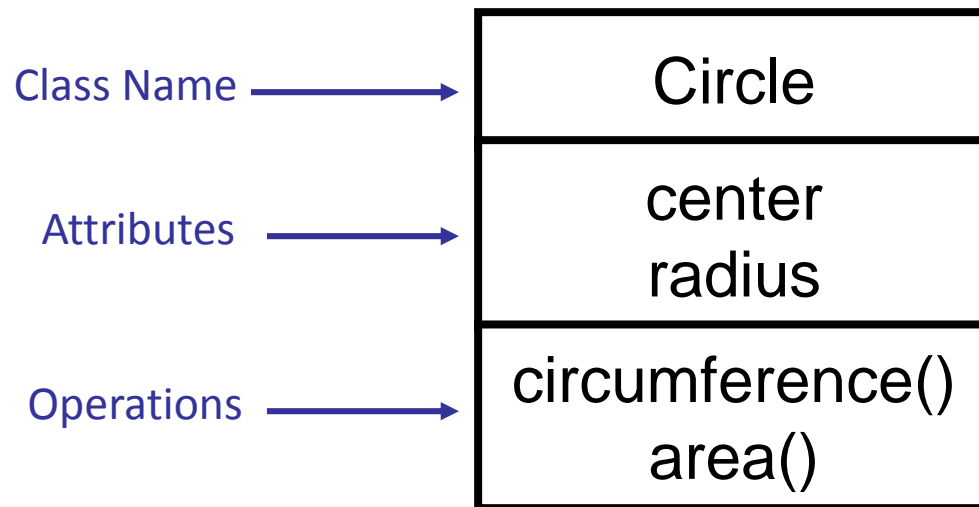
- **Defining Classes, Objects and Methods**
- **Accessor and Mutator Methods**
- **Constructors**
- **Static Members**
- **Wrapper Classes**
- **Parameter Passing**
- **Delegation**

Class and Method Definitions

- Java program consists of objects
 - Objects of class types
 - Objects that interact with one another
- Program objects can represent
 - Objects in real world
 - Abstractions
 - Software components

Java Classes

- A class is a collection of fields (data) and methods (procedure or function) that operate on that data.



Defining a Java Class

■ Syntax:

```
class  ClassName{  
    [fields declaration]  
    [methods declaration]  
}
```

■ Bare bone class definition:

```
/* This is my first java class.  
It is not complete yet. */  
class Circle {  
    // fields will come here  
    // methods will come here  
}
```

Adding Fields to Class Circle

■ Add fields

```
class Circle {  
    public double x, y; // center coordinates  
    public double r;    // radius of the circle  
}
```

- The fields are also called the *instance* variables.
 - Each object, or instance, of the class has its own copy of these instance variables
- Do not worry about what *public* means at this moment.
 - Access modifiers (public, private and protected will be covered next weeks)

Adding Methods to a Class

- A class with only data fields has no life.
 - Objects created by such a class **cannot respond to any messages**.
- Methods are declared inside the body of the class.
- The general form of a method declaration is:

```
type MethodName (parameter-list)
{
    Method-body;
}
```

- MethodName(parameter-list) part of the declaration is also known as the method signature.

Method signatures in a class must be unique!

Adding Methods to Class Circle

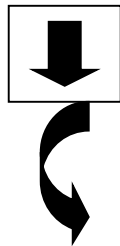
```
public class Circle {  
    public double x, y; // center of the circle  
    public double r;    // radius of the circle  
  
    // Method to return circumference  
    public double circumference() {  
        return 2 * 3.14 * r;  
    }  
  
    // Method to return area  
    public double area() {  
        return 3.14 * r * r;  
    }  
}
```


Defining Objects of a Class

- A class can be thought as a type
- A variable (object) can be defined as of that type (class)

```
Circle circleA, circleB;
```

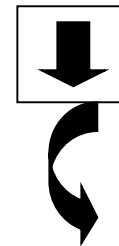
circleA



null

Points to nothing (Null Reference)

circleB



null

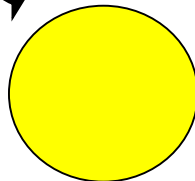
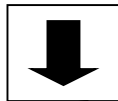
Points to nothing (Null Reference)

Creating Objects of a Class

- Objects are created by using the **new** keyword

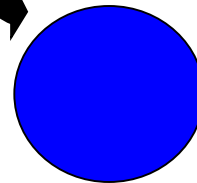
```
Circle circleA;  
circleA = new Circle();  
  
Circle circleB = new Circle();
```

circleA



Two different
circle objects!

circleB

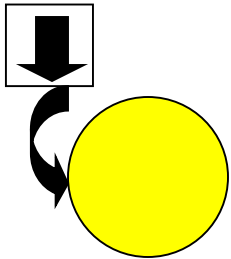


Creating Objects of a Class

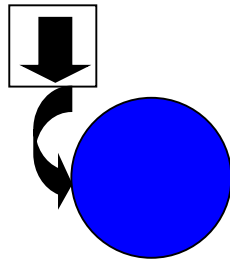
```
circleA = new Circle();  
circleB = new Circle();  
circleA = circleB;
```

Before Assignment

circleA

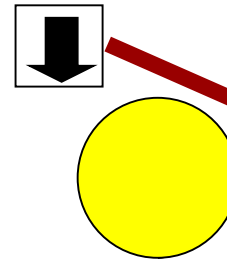


circleB

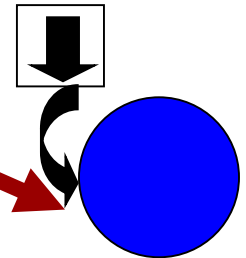


After Assignment

circleA



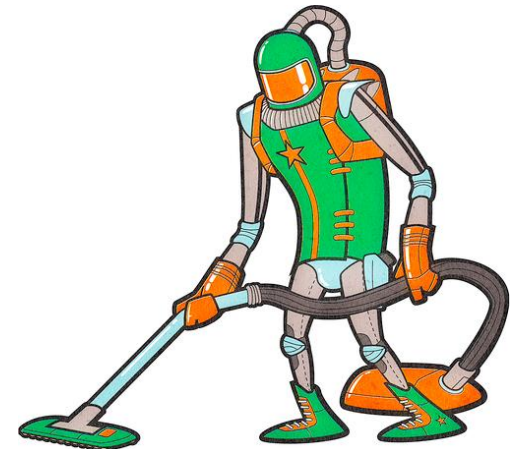
circleB



This object does not have a
reference anymore: **inaccessible!**

Garbage Collection

- The object which does not have a reference cannot be used anymore.
- Such object becomes a candidate for automatic garbage collection.
- Java collects garbage periodically and releases the memory occupied by such objects to be used in the future.



Using Objects

- Object's data is accessed by using the dot notation

```
Circle circleA = new Circle();  
  
circleA.x = 25.0;  
circleA.y = 25.0;  
circleA.r = 3.0;
```

- Object's methods are invoked by sending messages

```
double area = circleA.area();
```

A Complete Circle Class

```
public class Circle {
    public double x, y; // center of the circle
    public double r;    // radius of the circle

    // Methods to return circumference and area
    public double circumference() {
        return 2 * 3.14 * r;
    }
    public double area() {
        return 3.14 * r * r;
    }
    public static void main(String[] args) {
        Circle circleA = new Circle();
        circleA.x = 25.0;
        circleA.y = 25.0;
        circleA.r = 3.0;

        double area = circleA.area();
        System.out.println("Area of the circle is " + area);
    }
}
```

Class Files and Separate Compilation

- Each Java class definition usually in a file by itself.
 - File begins with name of the class
 - Ends with `.java`
- Class can be compiled separately.

```
public class Dog {
    public String name;    // Instance variables
    public String breed;
    public int age;

    // Method that returns nothing: void method
    public void writeOutput() {
        System.out.println("Name: " + name);
        System.out.println("Breed: " + breed);
        System.out.println("Age in calendar years: " + age);
        System.out.println("Age in human years: " +
                               getAgeInHumanYears());
    }

    // Method that returns a value
    public int getAgeInHumanYears() {
        int humanAge = 0;
        if (age <= 2) {
            humanAge = age * 11;
        } else {
            humanAge = 22 + ((age - 2) * 5);
        }
        return humanAge;
    }
}
```

Dog
+ name: String + breed: String + age : int
+ writeOutput(): void + getAgeInHumanYears(): int

How Old Is My Dog in Human Years

Size of Dog	Small Miniature Pinscher  20 lbs. or less	Medium Border Collie  21 - 50 lbs.
	Age of Dog	Age in Human Years
	1 Year	15
	2	24
	3	28
	4	32
	5	36
	6	40
	7	44
	8	48

Example Dog Class


```
public class DogDemo {  
    public static void main(String[] args) {  
        Dog balto = new Dog();  
        balto.name = "Balto";  
        balto.age = 8;  
        balto.breed = "Siberian Husky";  
        balto.writeOutput();  
  
        Dog scooby = new Dog();  
        scooby.name = "Scooby";  
        scooby.age = 42;  
        scooby.breed = "Great Dane";  
        System.out.println(scooby.name + " is a " + scooby.breed + ".");  
        System.out.print("He is " + scooby.age + " years old, or ");  
  
        int humanYears = scooby.getAgeInHumanYears();  
        System.out.println(humanYears + " in human years.");  
    }  
}
```

**DogDemo class contains
only a main method.**

Name: Balto
Breed: Siberian Husky
Age in calendar years: 8
Age in human years: 52

Program's output

Scooby is a Great Dane.
He is 42 years old, or 222 in human years.

```
public class Dog {  
    public String name;  
    public String breed;  
    public int age;  
  
    public void writeOutput() {  
        // method body  
    }  
  
    public int getAgeInHumanYears() {  
        // method body  
    }  
  
    public static void main(String[] args) {  
        Dog balto = new Dog();  
        balto.name = "Balto";  
        balto.age = 8;  
        balto.breed = "Siberian Husky";  
        balto.writeOutput();  
        ...  
    }  
}
```

Dog class could contain a
main method, too.

Multiple Classes in a Single File

```
class Computer {  
  
    void computer_method() {  
        System.out.println("Power gone! Shut down your PC soon...");  
    }  
  
    public static void main(String[] args) {  
        Computer my = new Computer();  
        Laptop your = new Laptop();  
  
        my.computer_method();  
        your.laptop_method();  
    }  
}  
  
class Laptop {  
    void laptop_method() {  
        System.out.println("99% Battery available.");  
    }  
}
```

The file *Computer.java* contains two class definitions.

```
$ javac Computer.java  
// will generate Computer.class and Laptop.class files.
```

Accessor and Mutator Methods

- A public method that returns data from a private instance variable is called an accessor method, a get method, or a getter.
 - The names of accessor methods typically begin with **get**.
- A public method that changes the data stored in one or more private instance variables is called a mutator method, a set method, or a setter.
 - The names of mutator methods typically begin with **set**.

Circle Class with Getters/Setters

```
public class Circle {  
    public double x, y; // center of the circle  
    public double r;    // radius of the circle  
  
    public double getX() { return x; }  
    public void setX(double centerX) { x = centerX; }  
    public double getY() { return y; }  
    public void setY(double centerY) { y = centerY; }  
    public double getR() { return r; }  
    public void setR(double radius) { r = radius; }  
  
    // Methods to return circumference and area  
    ...  
}
```

Constructors

- Constructor is a special method that gets invoked “automatically” at the time of object creation.
- Constructor is normally used for initializing objects with default values unless different values are supplied.
- Constructors **have the same name as the class name.**
- Constructor cannot return values.
- A class can have more than one constructor as long as they have different signature (i.e., different input arguments syntax).

Circle Class with Constructor

```
public class Circle {  
    public double x, y; // center of the circle  
    public double r;    // radius of the circle  
  
    // Constructor  
    public Circle(double centerX, double centerY, double radius) {  
        x = centerX;  
        y = centerY;  
        r = radius;  
    }  
  
    // Methods to return circumference and area  
    ...  
}
```

```
Circle aCircle = new Circle(10.0, 20.0, 5.0);
```

Multiple Constructors

- Sometimes we may want to initialize in a number of different ways, depending on circumstance.
- This can be supported by having multiple constructors having different input arguments (signatures).

Circle Class with Multiple Constructors

```
public class Circle {
    public double x, y; // center of the circle
    public double r;    // radius of the circle

    // Constructor
    public Circle(double centerX, double centerY, double radius) {
        x = centerX;
        y = centerY;
        r = radius;
    }

    public Circle(double radius) {
        x = 0; y = 0; r = radius;
    }

    public Circle() {
        x = 0; y = 0; r = 1.0;
    }

    // Methods to return circumference and area
    ...
}

Circle aCircle = new Circle(10.0, 20.0, 5.0);
Circle bCircle = new Circle(5.0);
Circle cCircle = new Circle();
```

Default and No-Argument Constructors

- Every class must have at least one constructor
 - If no constructors are declared, the compiler will create a default constructor
 - Takes no arguments and initializes instance variables to their initial values specified in their declaration or to their default values
 - Default values are **zero** for primitive numeric types, **false** for **boolean** values and **null** for references

Common Programming Error

- If a class has constructors, but none of the `public` constructors are no-argument constructors, and a program attempts to call a no-argument constructor to initialize an object of the class, a compilation error occurs.
- A constructor can be called with no arguments only if the class does not have any constructors (in which case the default constructor is called) or if the class has a `public` no-argument constructor.

The Keyword **this**

- **this** keyword can be used to refer to the object itself.
- It is generally used for accessing class members (from its own methods) when they have the same name as those passed as arguments.

```
public class Circle {  
    public double x, y; // center of the circle  
    public double r;    // radius of the circle  
  
    public double getX() { return x; }  
    public void setX(double x) { this.x = x; }  
    public double getY() { return y; }  
    public void setY(double y) { this.y = y; }  
    public double getR() { return r; }  
    public void setR(double r) { this.r = r; }  
  
    // Methods to return circumference and area  
    ...  
}
```

Static Variables

- Java supports definition of global variables that can be accessed without creating objects of a class.
 - Such members are called Static members.
- This feature is useful when we want to create a variable common to all instances of a class.
- One of the most common example is to have a variable that could keep a count of how many objects of a class have been created.
- Java creates only one copy for a static variable which can be used even if the class is never instantiated.

Using Static Variables

- Define the variable by using the **static** keyword

```
public class Circle {  
    // Class variable, one for the Circle class.  
    // To keep number of objects created.  
    public static int numCircles;  
  
    // Instance variables, one for each instance  
    // of the Circle class.  
    public double x,y,r;  
  
    // Constructor  
    Circle (double x, double y, double r){  
        this.x = x;  
        this.y = y;  
        this.r = r;  
        numCircles++;  
    }  
}
```

```
Circle circleA = new Circle(10, 12, 20);  
// numCircles = 1  
Circle circleB = new Circle(5, 3, 10);  
// numCircles = 2
```

Instance vs. Static Variables

- *Instance variables*: One copy per object. Every object has its own instance variables.
 - e.g. `x, y, r` (center and radius of the circle)
- *Static variables*: One copy per class.
 - e.g. `numCircles` (total number of circle objects created)

Static Methods

- A class can have methods that are defined as **static**.
- Static methods can be accessed without using objects. Also, there is **NO need to create objects**.
- Static methods are generally used to group related library functions that don't depend on data members of its class.
 - e.g., Math library functions.

Using Static Methods

```
class Comparator {  
    public static int max(int a, int b) {  
        if (a > b)  
            return a;  
        else  
            return b;  
    }  
  
    public static String max(String a, String b) {  
        if (a.compareTo(b) > 0)  
            return a;  
        else  
            return b;  
    }  
}
```

```
// Max methods are directly accessed using ClassName.  
// NO Objects created.  
System.out.println(Comparator.max(5, 10));  
System.out.println(Comparator.max("ANKARA", "SAMSUN"));
```

More Static Methods: The **Math** Class

- It is like including libraries in other languages
- It contains standard mathematical methods
 - They are all static
 - `Java.lang.Math`

```
Math.pow(2.0, 3.0)    // 8  
Math.max(5, 6)        // 6  
Math.round(6.2)       // 6  
Math.sqrt(4.0)        // 2.0
```

Object Cleanup (Destructor)

- Recall: Memory deallocation is automatic in Java
 - No dangling pointers and no memory leak problem.
- Java allows to define **finalize** method, which is invoked (if defined) just before the object destruction.
- This presents an opportunity to perform record maintenance operation or clean up any special allocations made by the user.
- The finalize method will be called by the Garbage Collector, but when this will happen is not deterministic.

```
protected void finalize() throws IOException {  
    Circle.numCircles = Circle.numCircles--;  
    System.out.println("Number of circles:" + Circle.num_circles);  
}
```

Wrapper Classes

- Each of Java's primitive data types has a class dedicated to it.
 - Boolean, Byte, Character, Integer, Float, Double, Long, Short
 - These are known as wrapper classes, because they "wrap" the primitive data type into an object of that class.
 - They contain useful predefined constants and methods
 - The wrapper classes are part of the `java.lang` package, which is imported by default into all Java programs.
 - Since Java 5.0 we have autoboxing and unboxing.

```
// Defining objects of wrapper class
Integer x = new Integer(33);
Integer y = 33; // Autoboxing
Integer yInt = y;

// Convert string to an integer
String s = "123";
int i = Integer.parseInt(s);

//Converting from hexadecimal to decimal
Integer hex2Int = Integer.valueOf("D", 16);
```

Parameter Passing

- Java works as «Call by Value» for parameter-passing.
 - Copy of the primitive types
 - Copy of the reference of the Class types.
- Copy of the reference to the object is passed into the method, original value unchanged, but you may change the attributes of the objects.

```

public class ReferenceTest {

    public static void main (String[] args){
        Circle c1 = new Circle(5, 5, 20);
        Circle c2 = new Circle(1, 1, 10);
        System.out.println ( "c1 Radius = " + c1.getRadius());
        System.out.println ( "c2 Radius = " + c2.getRadius());

        parameterTester(c1, c2);

        System.out.println ( "c1 Radius = " + c1.getRadius());
        System.out.println ( "c2 Radius = " + c2.getRadius());
    }

    public static void parameterTester(Circle circleA, Circle circleB){
        circleA.setRadius(15);
        circleB = new Circle(0, 0, 100);

        System.out.println ( "circleA Radius = " + circleA.getRadius());
        System.out.println ( "circleB Radius = " + circleB.getRadius());
    }

}

```

```

c1 Radius = 20.0
c2 Radius = 10.0
circleA Radius = 15.0
circleB Radius = 100.0
c1 Radius = 15.0
c2 Radius = 10.0

```

Delegation

- Ability for a class to delegate its responsibilities to another class.
- A way of making an object invoking services of other objects through containership.

Using Delegation

```
public class Point {  
    private double xCoord;  
    private double yCoord;  
  
    public double getXCoord() {  
        return xCoord;  
    }  
    public double getYCoord() {  
        return yCoord;  
    }  
}
```

```
public class Circle {  
    private Point center;  
    public double getCenterX() {  
        return center.getXCoord();           // Delegation  
    }  
    public double getCenterY() {  
        return center.getYCoord();           // Delegation  
    }  
}
```


Summary

- Classes, objects, and methods are the basic components used in Java programming.
- Constructors allow seamless initialization of objects.
- Classes can have static members, which serve as global members of all objects of a class.
- Objects can be passed as parameters and they can be used for exchanging messages.
- We will continue next week with encapsulation
 - which helps in protecting data from accidental or wrong usage and also offers better security for data.

Acknowledgments

- The course material used to prepare this presentation is mostly taken/adopted from the list below:
 - Java - An Introduction to Problem Solving and Programming, Walter Savitch, Pearson, 2012.
 - Rajkumar Buyya, University of Melbourne.