JAVA PROGRAMMING 1

Summer 2018 - Christian Hur

# Unit 3 Lecture - Classes, Methods, Objects

Readings: Chapter 3

# What Are Classes, Methods, Objects?

Understanding classes and how objects are instantiated (created) from them is the heart of object-oriented thinking. Let’s review and define some important jargons, terminologies, and methodologies in object-oriented programming.

### Classes

**Classes** are collections of properties and methods that are bound to objects, and they are the blueprints or templates for creating the objects.

Example:

class Student{

private String name;

private int age;

.

.

.

}

### Objects

**Objects** are the instances (or copies) created from the classes. Objects are the Nouns.

Example: Student s1 = new Student(); //a Student object called “s1”

### Methods

**Methods** are program modules that contain a series of statements that carry out a task. They are the behaviors or actions of the objects. Methods are the Verbs.

* **Mutator methods** set values. Also called Setter
* **Accessor methods** retrieve values. Also called Getter
* **Nonstatic methods**, those methods used with object instantiations, are called ***instance methods***.

### Properties

**Properties** are the attributes or characteristics of the objects. They describe the objects. Properties are sometimes called **Data Fields,** **Data Members**, or **Instance Variables**. Properties are the Adjectives.

Example:

private String name;

private int id;

private float gpa;

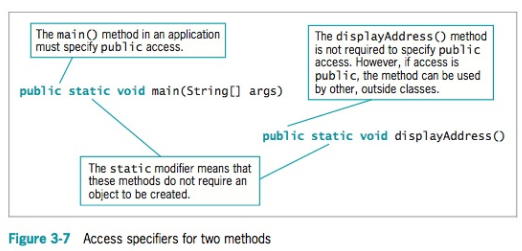
### Constructor

**Constructor** - A constructor is a method that establishes an object.

### Access modifier

**Access modifier** is sometimes used as another term for **access specifier**.

Examples: public, private, protected, default



### Object Instantiation

**Instantiation** is the process of creating objects from a class. An object is always instantiated with the “**new**” keyword in Java. For example:

|  |
| --- |
| //A concrete student class  class Student{  private String name;  private int age;  .  .  .  }  //A concrete Main project class  class MyProject {  public static void main(String[] args){  //Instantiation of a new Student object called “s1”  Student s1 = new Student();  //A second Student object “s2” is instantiated  Student s2 = new Student();  }  } |

|  |
| --- |
| More on these later…(Chapter 10) |
| **Inheritance** is a feature in which a class inherits or extends another class, thereby, acquiring the properties and methods of that class.  **Base class, parent class,** or **superclass** is a class that is used as a basis for inheritance. It’s the class that is inherited by another class (the child class).  **Derived class, subclass,** or **child class** is a class that inherits from a base class. It’s the class that extends or inherits another class (the parent class).  **Encapsulation** is information hiding. It refers to the enclosure or concealment of an object’s data (attributes and methods) from an outside source. The concept of a ***black box*** precisely describes this process in which one object cannot know the data of another object whether or not the objects are of the same type. A black box is a device you can use without understanding how it works.  **Polymorphism** is the technique of using the same method name to indicate different implementations. |

# Classes

A Java program **must** have at least one class, and it can have an unlimited number of classes. There are many kinds of classes in Java, but the three general types that all classes are classified under are:

1. **Concrete Class** - a concrete class is a class that can have instances. Objects are created directly from a concrete class. A concrete class cannot have any abstract methods.
2. **Abstract Class** - an abstract class is a class that cannot have direct instances (objects). Objects cannot be instanticated directly from it, but only through a subclass. An abstract class typically has at least an abstract method.
3. **Interface Class** - an interface is similar to that of an abstract class but it cannot have data fields and can only have abstract methods.

# Creating Classes

You create a class header with three parts:

* An optional access specifier (public, private, protected - classes are typicall set to **public**)
* The keyword **class**
* Any legal identifier you choose for the name of your class—starting with an uppercase letter is conventional

***Syntax:*** *blue text- required, [ ] - optional*

[access modifier] [abstract] **class identifier** [extends] [base class name] **{ }**

For example, a header for a class that represents an employee might be:

public class Employee {

}

A subclass that extends the Company base class:

public class Employee extends Company {

}

## 

# Data Fields

The body contains the data and methods for the class. The data components of a class are often referred to as **data fields** to help distinguish them from other variables you might use. Data fields aer always defined outside of any methods and within the class space and have class scope.

***Syntax****: blue text- required, [ ] - optional*

[access modifier] [non-access modifier] **data-type identifier**

public class Employee {

private int empNum; //instance variable

private static int ageLimit = 25; //static variable

}

# Instance Methods

Methods that are not defined as “static” are called instance methods. The method header is the first line of a method. It contains the following:

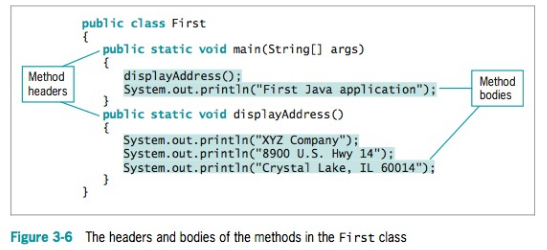
* Optional access specifiers
* A return type
* An identifier
* Parentheses
* Optional parameters

**Syntax**: *blue text- required, [ ] - optional*

[access modifier] [non-access modifier] **data-type identifier(** [parameters] **)** {

[return statement]; //must have a return statement if data-type is not “void”

}



Examples:

//no parameters and does not return anything

public static void main(String[] args){

//no return is required

}

//two parameters and returns a double

public double calculateGpa( double a, double b) {

return (a / b ); //returns a double

}

//parameter is an Employee object and returns a String

public String calculateRank( Employee emp1) {

if (emp1.gpa == 4.0)

return “Validictorian”); //returns this String

else if(emp1.gpa >= 3.5)

return “Top 10”; //or this String

else

return “None”;

}

## Mutators and Accessor Methods (aka Setters and Getters)

Methods that set or change field values are called mutator methods; methods that retrieve values are called accessor methods. In Java, mutator methods conventionally start with the prefix set, and accessor methods conventionally start with the prefix get.

public void setEmpNum(int emp) {

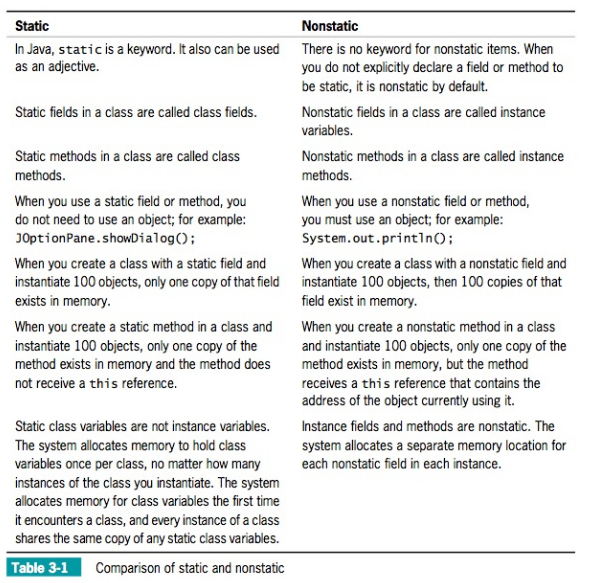
empNum = emp;

}

public int getEmpNum() {

return empNum;

}



# Creating Objects

Objects are created in two steps:

1. Define the type and identifier:   
  
 Employee emp1; //no memory allocation

2. Instantiate the new object with the “new” operator:

emp1 = new Employee(); //memory allocated

This is typically done in a single statement:

Employee emp1 = new Employee(); //memory allocated

# Constructors

A constructor is a special type of method that creates and initializes objects. When an object is created with the “new” operator, a constructor is always called to create the object. Objects cannot be created without a contructor. Every class must have a constructor. If you don’t supply one, Java will create a generic constructor (empty constructor) for it automatically. A constructor must have the same name as the class name. A class can have any number of constructors but each must have a unique ***signature*** (i.e. unique parameters).

**Syntax:** *blue text- required, [ ] - optional*

[access modifier] **class-name (** [parameters] **) { }**

Examples:

public class Employee {

private int empNum; //instance variable

private static int ageLimit; //static variable

// \*\* Default generic constructor is automatically created by Java

// if none is created by programmer

public Employee(){ }

// \*\* Three custom constructors with different signatures \*\*

// A constructor with no paramenters should be explicitly created to

// prevent accidental instantiation objects with default values

//initialize an instance variable - no parameters (default)

public Employee(){

this.empNum = 12;

}

//constructor with one parameter - different signature

public Employee(int a){

this.empNum = a;

}

public Employee(String a){

this.name = a;

}

//constructor with two parameters - another different signature

public Employee(int a, int b){

this.empNum = a;

this.ageLimit = b;

}

}

Examples of how the above constructors are used:

public static void main(String[] args){

Employee emp1 = new Employee(); //no arguments

Employee emp2 = new Employee(90); //one argument

Employee emp3 = new Employee(55,24); //two arguments

emp1.ageLimit = 21;

}

# Objects and Classes

*A class defines the properties and behaviors for objects.*

*Object-oriented programming (OOP) involves programming using objects. An object represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.*

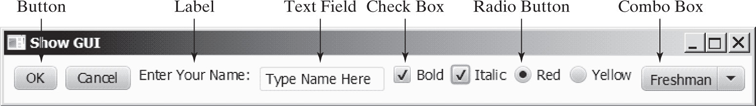
*An object has a unique* ***identity****,* ***state****, and* ***behavior****.*

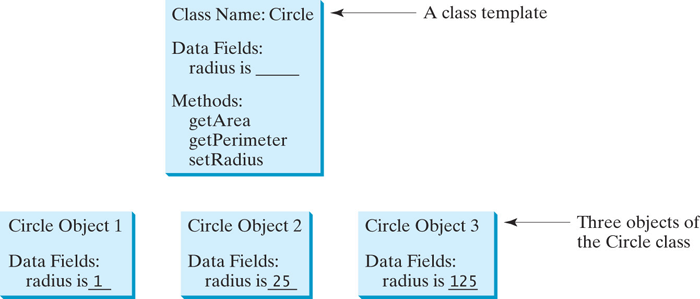
*The state of an object (also known as its properties or attributes) is represented by data fields with their current values.*

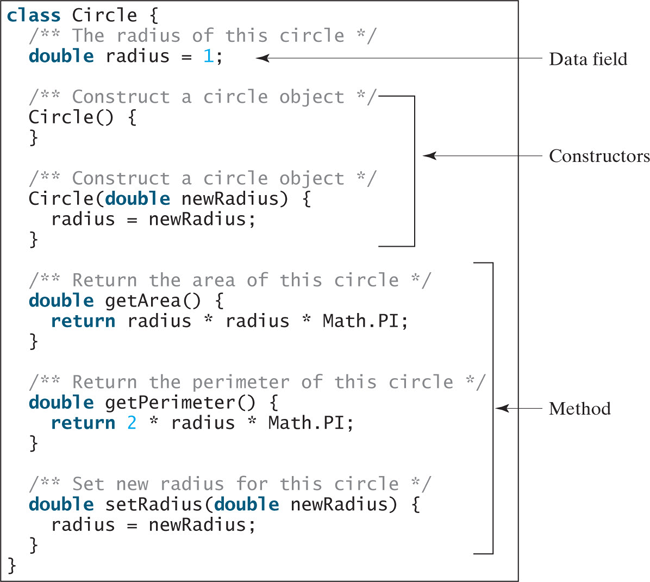
*The* ***behavior*** *of an object (also known as its actions) is defined by methods.*

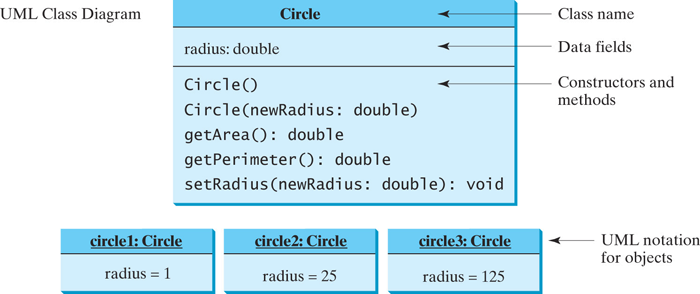
*A* ***class*** *is a* ***template****, blueprint, or contract that defines what an object’s data fields and methods will be. An* ***object*** *is an* ***instance*** *of a class.*

All of the components you see in the pictorial below are ***objects*** - an instance of a specific class (e.g. Button, Label, TextField, CheckBox, RadioButton, ComboBox).

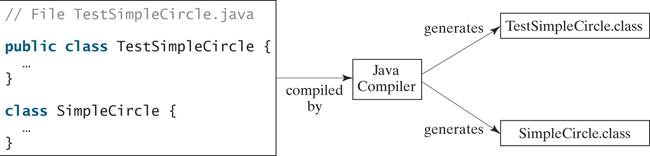


****





Each class in the source code file is compiled into a **.class** file (bytecode).



## Data Field Encapsulation

*Making data fields private protects data and makes the class easy to maintain.*

*A private data field cannot be accessed by an object from outside the class that defines the private field. However, a client often needs to retrieve and modify a data field. To make a private data field accessible, provide a* ***getter method to return its value****. To enable a private data field to be updated, provide a* ***setter method to set a new value****.*

* *A getter method is also referred to as an accessor*
* *A setter method is referred to as a mutator.*

*Getter: returnType*

*public returnType getPropertyName()*

/\*\* Return radius \*/  
 public double getRadius() {  
 return radius;  
 }

*Setter: void*

*public void setPropertyName(dataType propertyValue)*

/\*\* Set a new radius \*/  
 public void setRadius(double newRadius) {  
 radius = (newRadius >= 0) ? newRadius : 0;  
 }

## Inheritance and Polymorphism (Overview)

*Object-oriented programming allows you to define new classes from existing classes. This is called inheritance.*

*Inheritance enables you to define a general class (i.e., a superclass) and later extend it to more specialized classes (i.e., subclasses).*

*The keyword* ***super*** *refers to the superclass and can be used to invoke the superclass’s methods and constructors.*

*Example:*

public [[arrow]]CircleFromSimpleGeometricObject(  
 double radius, String color, boolean filled) {  
 super(color, filled);  
 this.radius = radius;  
}

1 public class Faculty extends Employee {  
 2 public static void main(String[] args) {  
 3 new Faculty();  
 4 }  
 5   
 6 public Faculty() {  
 7 System.out.println("(4) Performs Faculty's tasks");  
 8 }  
 9 }  
10   
11 class Employee extends Person {  
12 public Employee() {  
13 this("(2) Invoke Employee's overloaded constructor");  
14 System.out.println("(3) Performs Employee's tasks ");  
15 }  
16   
17 public Employee(String s) {  
18 System.out.println(s);  
19 }  
20 }  
21   
22 class Person {  
23 public Person() {  
24 System.out.println("(1) Performs Person's tasks");  
25 }  
26 }