Lecture 6

OOP

- OOP stands for Object-Oriented Programming.
- Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Advantages of OOP

- Object-oriented programming has several advantages over procedural programming:
  - OOP is faster and easier to execute
  - OOP provides a clear structure for the programs
  - OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
  - OOP makes it possible to create full reusable applications with less code and shorter development time

OOP topics

- **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts.
  - Object
  - Class
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation

OOP design

- Coupling
- Cohesion
- Association
- Aggregation
- Composition

Class

- Classes are the single most important language feature that facilitates **object-oriented programming (OOP)**, the dominant programming methodology in use today.

- Pre-defined class: Scanner, String

- An object is a value of a class type and is referred to as an instance of the class. An object differs from a value of a primitive type in that it has **methods** (actions) as well as **data**.

Classes and Objects

- **Classes** and **objects** are the two main aspects of object-oriented programming.

- A **class** is a template for objects, and an **object** is an instance of a class.

- When the individual **objects** are created, they inherit all the variables and methods from the **class**.

What is an object in Java

- An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible).

State: represents the data (value) of an object.

Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.

Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Object

- **An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

- Object Definitions:
  - An object is a real-world entity.
  - An object is *a runtime entity*.
  - The object is an entity which has state and behavior.
  - The object is an instance of a class.

Java Classes/Objects

- Java is an object-oriented programming language.
- Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

What is a class in Java

- A class in Java can contain
  - Fields
  - Methods
  - Constructors
  - Blocks
  - Nested class and interface

Syntax of class

- Syntax of class

```
class <class_name>{
    field;
    method;
}
```

Create a Class

- To create a class, use the keyword class:

```
public class Main {
    int x = 5;
}
```

Class Attributes

- Create a class called "Main" with two attributes: x and y

```
public class Main {
    int x = 5;
    int y = 3;
}
```

Accessing Attributes

- Attributes can be accessed by creating an object of the class, and by using the dot syntax (.):

```
public class Main {
    int x = 5;
    public static void main(String[] args) {
        Main myObj = new Main();
        System.out.println(myObj.x);
    }
}
```

Modify Attributes

- If you don't want the ability to override existing values, declare the attribute as final

```
public class Main {
    final int x = 10;
    public static void main(String[] args) {
        Main myObj = new Main();
        myObj.x = 25; // will generate an error:
        // cannot assign a value to a final variable
        System.out.println(myObj.x);
    }
}
```

Multiple Attributes

- You can specify as many attributes as you want:

```
public class Main {
    String fname = “John";
    String lname = “Doe";
    int age = 24;

    public static void main(String[] args) {
        Main myObj = new Main();
        System.out.println("Name: " + myObj.fname + " " +
        myObj.lname);
        System.out.println("Age: " + myObj.age);
    }
}
```

Create an Object

- In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

- To create an object of Main, specify the class name, followed by the object name, and use the keyword new:

```
public class Main {
    int x = 5;
    public static void main(String[] args) {
        Main myObj = new Main();
        System.out.println(myObj.x);
    }
}
```

The **new** Operator

- The new operator is used to create an object of a class and associate the object with a variable that names it.

- SYNTAX: Class_Variable = new Class_Name( );

- EXAMPLE:

```
DateFirstTry date;
date = new DateFirstTry();
```

which is usually written in the following equivalent form:

```
DateFirstTry date = new DateFirstTry();
```

Multiple Objects

- You can create multiple objects of one class:

```
public class Main {
    int x = 5;
    public static void main(String[] args) {
        Main myObj1 = new Main();  // Object 1
        Main myObj2 = new Main();  // Object 2
        System.out.println(myObj1.x);
        System.out.println(myObj2.x);
    }
}
```

Multiple Classes

- You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

- Remember that the name of the java file should match the class name.

Multiple Classes

```
// Main.java
public class Main {
    int x = 5;
}
```

```
// Second.java
class Second {
    public static void main(String[] args) {
        Main myObj = new Main();
        System.out.println(myObj.x);
    }
}
```

Create an object

- There are many ways to create an object in java.
- By new keyword
- By newInstance() method
- By clone() method
- By deserialization
- By factory method etc

Multiple Objects

- If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

```
public class Main {
    int x = 5;
    public static void main(String[] args) {
        Main myObj1 = new Main();  // Object 1
        Main myObj2 = new Main();  // Object 2
        myObj2.x = 25;
        System.out.println(myObj1.x);  // Outputs 5
        System.out.println(myObj2.x);  // Outputs 25
    }
}
```

Initialize object

- There are 3 ways to initialize object in Java.
  - By reference variable
  - By method
  - By constructor

Anonymous object

- Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

- If you have to use an object only once, an anonymous object is a good approach.

```
new Calculation(); //anonymous object

new Calculation().fact(5);
```

Constructors

- A **constructor** is a variety of method that is called when an object of the class is created using **new**. Constructors are used to initialize objects.

- A **constructor** must have the same name as the class to which it belongs. Arguments for a constructor are given in parentheses after the class name, as in the following examples.

- A **constructor** is defined very much like any ordinary method except that it does not have a type returned and does not even include a void in the constructor heading.

Constructors

- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

- It is a special type of method which is used to initialize the object.

- Every time an object is created using the new() keyword, at least one constructor is called.

- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Constructors

- There are two types of constructors in Java: **no-arg constructor (default constructor)**, and **parameterized constructor**.

- It is called constructor because it constructs the values at the time of object creation.

- It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Type of Constructor

- Default Constructor: A constructor is called "Default Constructor" when it doesn't have any parameter.

- Parameterized Constructor: A constructor which has a specific number of parameters is called a parameterized constructor.

No-Argument Constructor

- A constructor with no parameters is called a **no-argument constructor**. If your class definition contains absolutely no constructor definitions, then Java will automatically create a no-argument constructor.

- If your class definition contains one or more constructor definitions, then Java does not automatically generate any constructor; in this case, what you define is what you get. Most of the classes you define should include a definition of a no-argument constructor.

Rule of Constructors

- There are three rules defined for the constructor.
  - Constructor name must be the same as its class name
  - A Constructor must have no explicit return type
  - A Java constructor cannot be abstract, static, final, and synchronized

Constructors

- A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

```
public class Main {
    int x;  // Create a class attribute
    // Create a class constructor for the Main class
    public Main() {
        x = 5;  // Set the initial value for the class attribute x
    }
    public static void main(String[] args) {
        Main myObj = new Main(); // Create an object of class Main (This
        will call the constructor)
        System.out.println(myObj.x); // Print the value of x
    }
}
```

Constructors

- Note that the constructor name must **match the class name**, and it cannot have a **return type** (like void).

- Also note that the constructor is called when the object is created.

- All classes have constructors by default: if you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.

Constructor Parameters

- Constructors can also take parameters, which is used to initialize attributes.

```
public class Main {
    int x;
    int y;
    public Main(int a, int b) {
        x = a;
        y = b;
    }
    public static void main(String[] args) {
        Main myObj = new Main(5,10);
        System.out.println(myObj.x);
        System.out.println(myObj.y);
    }
}
```

Is a Constructor Really a Method?

- There are differing opinions on whether or not a constructor should be called a method. Most authorities call a constructor a method but emphasize that it is a special kind of method with many properties not shared with other kinds of methods.

- Some authorities say a constructor is a method-like entity but not, strictly speaking, a method. All authorities agree about what a constructor is; the only disagreement is over whether or not it should be referred to as a method.Thus, this is not a major issue. However, whenever you hear a phrase such as “all methods,” you should make sure you know whether it does or does not include constructors.

- To avoid confusion, we try to use the phrase “constructors and methods” when we want to include constructors.

Constructor Overloading

- In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.
- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Copy Constructor

- There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

- There are many ways to copy the values of one object into another in Java. They are:
  - By constructor
  - By assigning the values of one object into another
  - By clone() method of Object class

Public vs Private

- **Private** members can only be accessed by other members of the same class.

- **Public** members can be accessed by any class, anywhere in the program.

- Once you label an instance variable as **private**, there is then no way to change its value (nor to reference the instance variable in any other way) except by using one of the methods belonging to the class. Note that even when an instance variable is private, you can still access it through methods of the class.

Method in Java

- In general, a **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using **methods**.

- In this section, we will learn **what is a method in Java, types of methods, method declaration,** and **how to call a method in Java**.

What is a method in Java?

- A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

- The most important method in Java is the **main()** method.

Class Methods

- myMethod() prints a text (the action), when it is **called**. To call a method, write the method's name followed by two parentheses **()** and a semicolon

```
public class Main {
    static void myMethod() {
        System.out.println("Hello World!”);
    }
    public static void main(String[] args) {
        myMethod();
    }
}
```

Modifiers

- The **public** keyword is an **access modifier**, meaning that it is used to set the access level for classes, attributes, methods and constructors.

- We divide modifiers into two groups:

  - **Access Modifiers** - controls the access level

  - **Non-Access Modifiers** - do not control access level, but provides other functionality

Classe Access Modifiers

Modifier	Description
public	The class is accessible by any other class
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier.

Access Modifiers for attributes, methods and constructors

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the declared class
protected	The code is only accessible in the same package. This is used when you don't specify a modifier.
default	The code is accessible in the same package and subclasses.

Non-Access Modifiers for class

Modifier	Description
final	The class cannot be inherited by other classes
abstract	The class cannot be used to create objects, to access an abstract class, it must be inherited from another class.

Non-Access Modifiers for attributes and methods

Modifier	Description
final	Attributes and methods cannot be overridden/modified
static	Attributes and methods belongs to the class, rather than an object
abstract	Can only be used in an abstract class, and can only be used on methods.
transient	Attributes and methods are skipped when serializing the object containing them
synchronized	Methods can only be accessed by one thread at a time
volatile	The value of an attribute is not cached thread-locally, and is always read from the "main memory"

Accessor & Mutator

- **Accessor Method:** The method(s) that reads the instance variable(s) is known as the accessor method. We can easily identify it because the method is prefixed with the word **get**. It is also known as **getters**. It returns the value of the private field. It is used to get the value of the private field.

- **Mutator Method:** The method(s) read the instance variable(s) and also modify the values. We can easily identify it because the method is prefixed with the word **set**. It is also known as **setters** or **modifiers**. It does not return anything. It accepts a parameter of the same data type that depends on the field. It is used to set the value of the private field.

Get and Set

- **private** variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public **get** and **set** methods.

- The **get** method returns the variable value, and the **set** method sets the value.

- Syntax for both is that they start with either **get** or **set**, followed by the name of the variable, with the first letter in upper case.

Encapsulation

- **Encapsulation** means that the data and the actions are combined into a single item (in our case, a class object) and that the details of the implementation are hidden.

- The terms information hiding and encapsulation deal with the same general principle: If a class is well designed, a programmer who uses a class need not know all the details of the implementation of the class but need only know a much simpler description of how to use the class.

Encapsulation

- The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users.

- declare class variables/attributes as **private**

- provide public **get** and **set** methods to access and update the value of a **private** variable

Encapsulation

```
// Person.java
public class Person {
	private String name; // private = restricted access
	// Getter
	public String getName() {
		return name;
	}
	// Setter
	public void setName(String newName) {
		this.name = newName;
	}
}
// Maiin.java
public class Main {
	public static void main(String[] args) {
		Person myObj = new Person();
		myObj.setName("John"); // Set the value of the name variable to “John"
		System.out.println(myObj.getName());
	}
}
```

Why Encapsulation

- Better control of class attributes and methods
- Class attributes can be made **read-only** (if you only use the **get** method), or **write-only** (if you only use the **set** method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

Static vs. Non-Static

- We often see Java programs that have either **static** or **public** attributes and methods.

- **static** method means that it can be accessed without creating an object of the class.

Static Method

- A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

- The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

Static Method

- A **static method** is one that can be used without a calling object. With a static method, you normally use the class name in place of a calling object. When you define a static method, you place the keyword static in the heading of the definition.

- Since it does not need a calling object, a static method cannot refer to an instance variable of the class, nor can it invoke a non static method of the class (unless it creates a new object of the class and uses that object as the calling object). Another way to phrase it is that, in the definition of a static method, you cannot use an instance variable or method that has an **implicit** or **explicit** this for a calling object.

Non-static Method

- The method of the class is known as an **instance method**. It is a **non-static** method defined in the class.

- Before calling or invoking the instance method, it is necessary to create an object of its class. Let's see an example of an instance method.

Static vs. Non-Static

```
public class Main {
    // Static method
    static void myStaticMethod() {
        System.out.println("Static methods can be called without creating objects”);
    }
    // Public method
    public void myPublicMethod() {
        System.out.println("Public methods must be called by creating objects”);
    }
    // Main method
    public static void main(String[] args) {
        myStaticMethod(); // Call the static method
        // myPublicMethod(); This would compile an error
        Main myObj = new Main(); // Create an object of Main
        myObj.myPublicMethod(); // Call the public method on the object
    }
}
```

Static Variables

- A static variable belongs to the class as a whole. All objects of the class can read and change the static variable. Static variables should normally be private, unless they happen to be defined constants.

- SYNTAX

```
private static Type Variable_Name;
private static Type Variable_Name = Initial_Value;
public static final Type Variable_Name = Constant_Value;
```

- EXAMPLES

```
private static String lastUser;
private static int turn = 0;
public static final double PI = 3.14159;
```

static keyword

- The **static keyword** in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

- The static can be:
  - Variable (also known as a class variable)
  - Method (also known as a class method)
  - Block
  - Nested class

static variable

- If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.

- The static variable gets memory only once in the class area at the time of class loading.

- Advantages of static variable: It makes your program **memory efficient**

static method

- If you apply static keyword with any method, it is known as static method.
- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Restrictions for the static method

- There are two main restrictions for the static method.

- The static method can not use non static data member or call non-static method directly.

- this and super cannot be used in static context.

- Why is the Java main method static?

- It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Block

- A **block** is another name for a compound statement that is, a list of statements enclosed in braces. However, programmers tend to use the two terms in different contexts. When you declare a variable within a compound statement, the compound statement is usually called a **block**.

- The variables declared in a block are local to the block, so these variables disappear when the execution of the block is completed. However, even though the variables are local to the block, their names cannot be used for anything else within the same method definition.

static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

Local vs Global variable

- **Local variable**: A variable declared within a method definition is called a local variable. If two methods each have a local variable of the same name, they are two different variables that just happen to have the same name.

- **Global variable**: Thus far, we have discussed two kinds of variables: instance variables, whose meaning is confined to an object of a class, and local variables, whose meaning is confined to a method definition. Some other programming languages have another kind of variable called a global variable, whose meaning is confined only to the program. **Java does not have these global variables**.

this keyword

- 6 usage of java this keyword
- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

this: refer current class

- The **this** keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

- Within a method definition, you can use the keyword **this** as a name for the calling object. If an instance variable or another method in the class is used without any calling object, then this is understood to be the calling object.

this: to invoke current class method

- You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method.

this(): to invoke current class constructor

- The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

- Calling default constructor from parameterized constructor

- The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors

- Call to this() must be the first statement in constructor.

this: to pass as an argument in the method

- The this keyword can also be passed as an argument in the method. It is mainly used in the event handling.

- In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

this: to pass as argument in the constructor call

- We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes.

this: can be used to return current class instance

- We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive).

Wrapper Classes

- Every primitive type has a corresponding **wrapper class**. A wrapper class allows you to have a class object that corresponds to a value of a primitive type. Wrapper classes also contain a number of useful predefined constants and static methods.

- The process of going from a value of a primitive type to an object of its wrapper class is sometimes called **boxing**.

- This process of going from an object of a wrapper class to the corresponding value of a primitive type is sometimes called **unboxing**.

Static Methods in Wrapper Classes

- The material on wrapper classes that we have seen thus far explains why they are called wrapper classes. However, possibly more importantly, the wrapper classes contain a number of useful constants and static methods.

- So, wrapper classes have two distinct personalities: One is their ability to produce class objects corresponding to values of primitive types, and the other is as a repository of useful constants and methods.

Variable and memory

- A computer has two forms of memory called secondary memory and main memory.

- The **secondary memory** is used to hold files for more or less permanent storage.

- The **main memory** is used by the computer when it is running a program.

Byte and Address

- Main memory is divided into numbered locations called **bytes**. The number associated with a byte is called its **address**.

- A group of consecutive bytes is used as the location for the value of a variable. The address of the first byte in the group is used as the address of this larger memory location.

8 bits = 1 byte

- A **byte** is a memory location that can hold 8 bits. What is so special about 8? Why not 10 bits? There are two reasons why 8 is special.

- **First**, 8 is a power of 2 (8 is 2^3). Since computers use bits, which have only two possible values, powers of 2 are more convenient than powers of 10.

- **Second**, it turns out that 7 bits are required to code a single character of the ASCII character set. So 8 bits (1 byte) is the smallest power of 2 that will hold a single ASCII character.

Variables of a Class Type Hold References

- A variable of a primitive type stores a value of that type. However, a variable of a class type does not store an object of that class.

- A variable of a class type stores the reference (memory address) of where the object is located in the computer's memory.

- This causes some operations, such as = and ==, to behave quite differently for variables of a class type than they do for variables of a primitive type.

- A type whose variables contain references are called **reference types**. In Java, class types are reference types, but primitive types are not reference types.

Assignment with variables of a class type

- Note that when you use the assignment operator with variables of a class type, you are assigning a reference (memory address), so the result of the following is to make **variable1** and **variable2** two names for the same object:
    - variable2 = variable1;
- A variable of a class type stores a memory address, and a memory address is a number. However, a variable of a class type cannot be used like a variable of a number type, such as int or double. This is intentional. The important property of a memory address is that it identifies a memory location. The fact that the implementors used numbers, rather than letters or strings or something else, to name memory locations is an accidental property. Java prevents you from using this accidental property to stop you from doing things such as obtaining access to restricted memory or otherwise screwing up the computer.