Lecture 7

import Statement

- You can use a class from a package in any program or class definition by placing an **import statement** that names the package and the class from the package at the start of the file containing the program (or class definition). The program (or class definition) need not be in the same directory as the classes in the package.

- Syntax: import Package_Name.Class;

- You can import all the classes in a package by using an asterisk in place of the class's name.

- Syntax: import Package_Name.*;

import Statement

- This package statement should be at the **beginning** of the file. Only blank lines and comments may precede the package statement.

- If there are both import statements and package statements, any package statements come **before** the import statements.

- Aside from the addition of the package statement, class files are just as we have already described them.

Java Packages

- A **package** is a collection of classes that have been grouped together into a directory and given a package name. The classes in the package are each placed in a separate file, and the file is given the same name as the class, just as we have been doing all along. Each file in the package must have the following at the beginning of the file. Only comments and blank lines may precede this package statement.

- Syntax: package Package_Name;

- Example: package java.util;

Java Packages

- A **package** is Java's way of forming a library of classes. You can make a package from a group of classes and then use the package of classes in any other class or program you write without the need to move the classes to the directory (folder) in which you are working.

- All you need to do is include an **import** statement that names the package.

Package Names and the CLASSPATH Variable

- A package name must be a path name for the directory that contains the classes in the package, but the package name uses dots in place of \ or / (whichever your operating system uses).

- When naming the package, use a relative path name that starts from any directory listed in the value of the CLASSPATH (environment) variable.

Java Packages

- A package in Java is used to group related classes. Think of it as **a folder in a file directory**. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

- Built-in Packages (packages from the Java API)

- User-defined Packages (create your own packages)

Built-in Packages

- The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment. The library contains components for managing input, database programming, and much much more.

- The library is divided into **packages** and **classes**. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

- To use a class or a package from the library, you need to use the **import** keyword

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

Import a Class

- If you find a class you want to use, for example, the **Scanner** class, **which is used to get user input**

```
import java.util.Scanner;
```

Import a Package

- There are many packages to choose from. In the previous example, we used the **Scanner** class from the **java.util** package. This package also contains date and time facilities, random-number generator and other utility classes.

- To import a whole package, end the sentence with an asterisk sign (*****). The following example will import ALL the classes in the **java.util** package

```
import java.util.*;
```

User-defined Packages

- To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer.

- The **-d** keyword specifies the destination for where to save the class file. You can use any directory name, like c:/user (windows), or, if you want to keep the package within the same directory, you can use the dot sign "**.**", like in the example above.

- **Note:** The package name should be written in lower case to avoid conflict with class names.

Inheritance in Java

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Inheritance

- In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- **subclass** (child, derived) - the class that inherits from another class

- **superclass** (parent, base) - the class being inherited from

- To inherit from a class, use the **extends** keyword.

Inheritance

- Define a **derived class** by starting with another already defined class and adding (or changing) methods, instance variables, and static variables. The class you start with is called the **base class**.

- The derived class inherits all the public methods, all the public and private instance variables, and all the public and private static variables from the base class, and it can add more instance variables, more static variables, and more methods. So, of the things we have seen thus far, the only members not inherited are private methods.

- A derived class is also called a **subclass**, in which case the base class is usually called a **superclass**.

Inherited Members

- A derived class automatically has all the instance variables, all the static variables, and all the public methods of the base class. These members from the base class are said to be **inherited**.

- These inherited methods and inherited instance and static variables are, with one exception, not mentioned in the definition of the derived class, but they are automatically members of the derived class.

- The one exception is as follows: As explained in the subsection “Overriding a Method Definition,” you can give a definition for an inherited method in the definition of the derived class; this will redefine the meaning of the method for the derived class.

Parent and Child Classes

- A base class is often called the **parent class**. A derived class is then called a **child class**. This analogy is often carried one step further.

- A class that is a parent of a parent of a parent of another class (or some other number of “parent of” iterations) is often called an **ancestor class**.

- If class A is an ancestor of class B, then class B is often called a **descendent** of class A.

Multi Inheritance

- To reduce the complexity and simplify the language, **multiple inheritance is not supported in java**.

- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Aggregation

- If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

- Why use Aggregation? For Code Reusability.

- When use Aggregation?

- Code reuse is also best achieved by aggregation when there is no is-a relationship.

- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

- If we have to perform only one operation, having same name of the methods increases the readability of the program.

- Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

Advantage of overloading

- Method overloading increases the readability of the program.
- Different ways to overload the method
- There are two ways to overload the method in java
  - By changing number of arguments
  - By changing the data type

Method Overloading

- Why Method Overloading is not possible by changing the return type of method only?

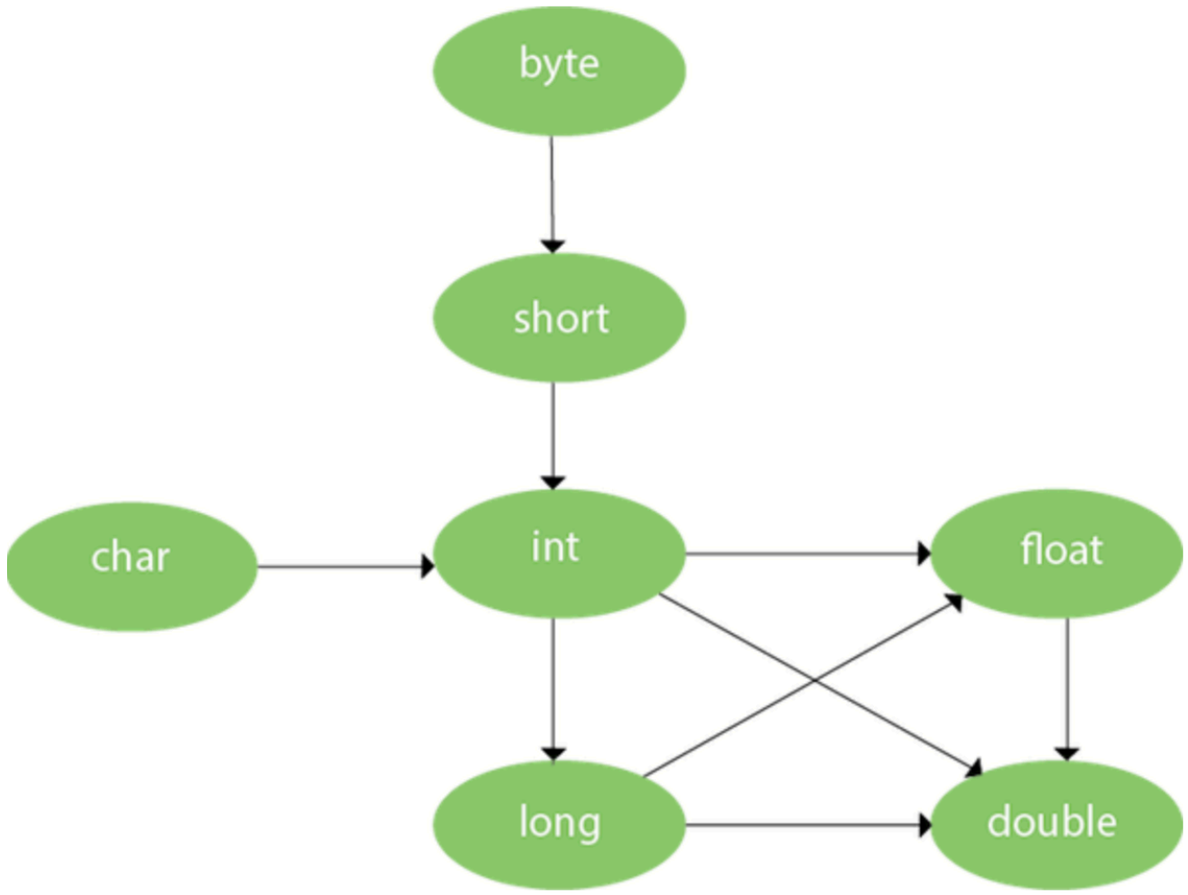
- In java, method overloading is not possible by changing the return type of the method only because of ambiguity.

- Can we overload java main() method?

- Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

Method Overloading and Type Promotion

- One type is promoted to another implicitly if no matching datatype is found.



Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

- In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Method Overriding

- Usage of Java Method Overriding
- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism
- Rules for Java Method Overriding
- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

Method Overriding

- Can we override static method?

- No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

- Why can we not override static method?

- It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

- Can we override java main method?

- No, because the main is a static method.

Covariant Return Type

- The covariant return type specifies that the return type may vary in the same direction as the subclass.

- Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type.

Advantages of Covariant Return Type

- Following are the advantages of the covariant return type.

1) Covariant return type assists to stay away from the confusing type casts in the class hierarchy and makes the code more usable, readable, and maintainable.

2) In the method overriding, the covariant return type provides the liberty to have more to the point return types.

3) Covariant return type helps in preventing the run-time *ClassCastExceptions* on returns.

Super Keyword

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

- Usage of Java super Keyword

  - super can be used to refer immediate parent class instance variable.

  - super can be used to invoke immediate parent class method.

  - super() can be used to invoke immediate parent class constructor.

Block

- **Instance Initializer block** is used to initialize the instance data member. It run each time when object of the class is created.

- The initialization of the instance variable can be done directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

- Why use instance initializer block?

- Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc.

Block

- Rules for instance initializer block :

- The instance initializer block is created when instance of the class is created.
- The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
- The instance initializer block comes in the order in which they appear.

The final Keyword

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be 1) variable 2) method 3) class
- The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

The final Keyword

- If you don't want other classes to inherit from a class, use the **final** keyword:

- If you try to access a **final** class, Java will generate an error

```
final class A {
  ...
}
class B extends A {
  ...
}
```