GLS UNIVERSITY

SEM – III

0301401 - CORE JAVA

Dr. Disha Shah
Prof. Vidhi Thakkar

Index

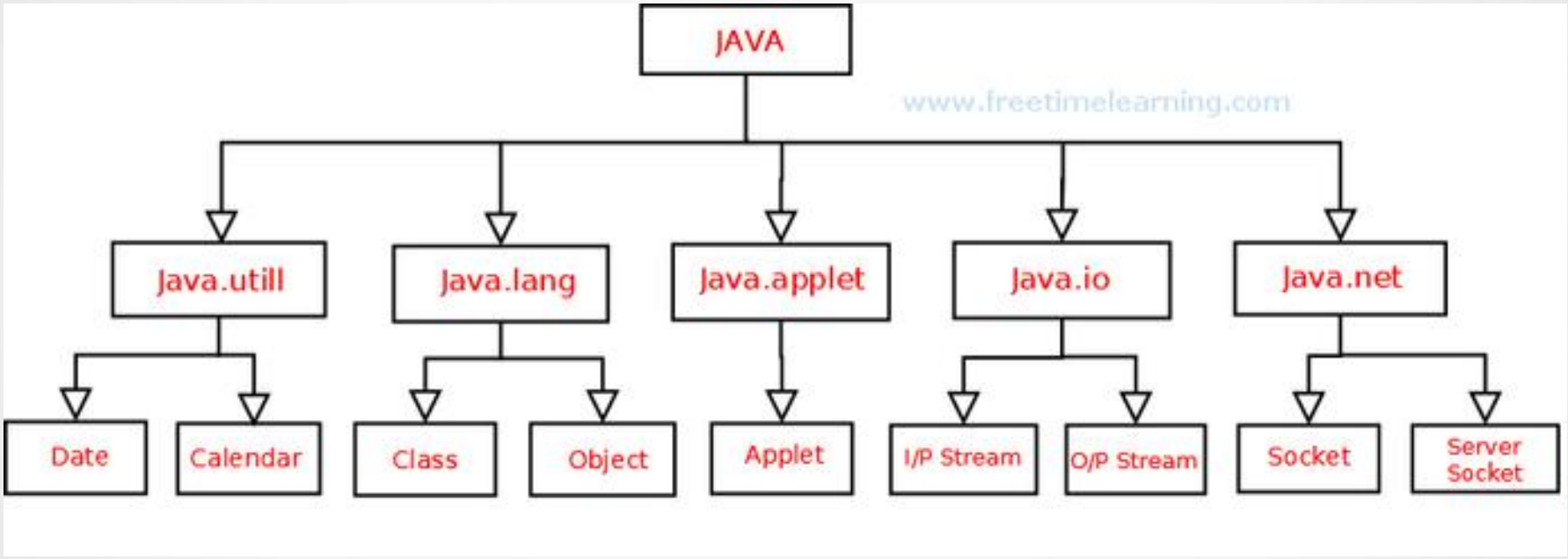
- Inheritance
- Interface
- Packages
  - Introduction
  - Creating Packages
  - Using Packages
  - Access Protection
  - Java.lang.package
    - Object class
    - Wrapper class
  - String class
  - StringBuffer class

Unit – 3
Packages in Java

Introduction

- A java package is a mechanism for organizing Java classes into groups.
- A java package is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form:
  - built-in package
  - user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- **Advantage of Java Package**
  - Java package is used to categorize the classes and interfaces so that they can be easily maintained.
  - Java package provides access protection.
  - Java package removes naming collision.

Introduction



Packages in Java

- A package declaration resides at the top of a Java source file.
- All source files to be placed in a package have common package name.
- A package provides a unique namespace for the classes it contains.
- A package can contain:
  - Classes
  - Interfaces
  - Enumerated types
  - Annotations
- Two classes in different packages can have the same name.
- Packages provide a mechanism to hide its classes from being used by programs or packages belonging to other classes.

Packages in Java

- **How to compile java package**

  javac -d directory javafilename

- For example

  javac -d . Simple.java

- The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

- **How to run java package program**

  To Compile: javac -d . Simple.java

  To Run: java mypack.Simple

  Output:Welcome to package

Packages in Java

How to access package from another package?

There are three ways to access the package from outside the package.

import packagename.*;

import package.classname;

fully qualified name.

1) Using packagename.*

- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- The import keyword is used to make the classes and interface of another package accessible to the current package.

Packages in Java

2) Using packagename.classname

- If you import package.classname then only declared class of this package will be accessible.

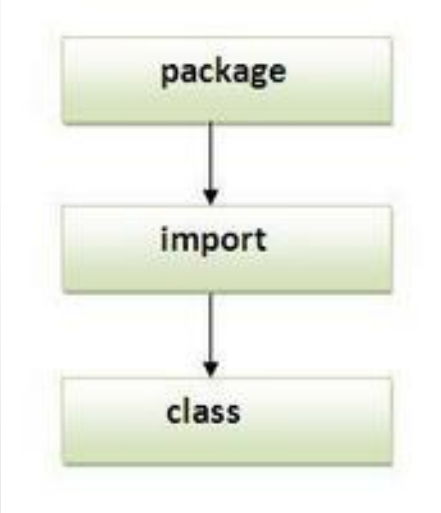
3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Subpackages in Java

- If you import a package, subpackages will not be imported.
- If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages.
- Hence, you need to import the subpackage as well.
- Sequence of the program must be package then import then class.



Subpackages in Java

- Package inside the package is called the subpackage. It should be created to categorize the package further.
- The standard of defining package is domain.company.package
- e.g. abc.java.bean

Access Modifiers in Java

- There are two types of modifiers in java:
  - access modifiers
  - non-access modifiers
- The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
- There are 4 types of java access modifiers:
  - Private
  - Default
  - Protected
  - Public
- There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc.

Access Modifiers in Java

Private access modifier:

- The private access modifier is accessible only within class.
- If you make any class constructor private, you cannot create the instance of that class from outside the class.
- A class cannot be private or protected except nested class.

Default access modifier:

- If you don't use any modifier, it is treated as default by default.
- The default modifier is accessible only within package.

Protected access modifier

- The protected access modifier is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Access Modifiers in Java

Public access modifier

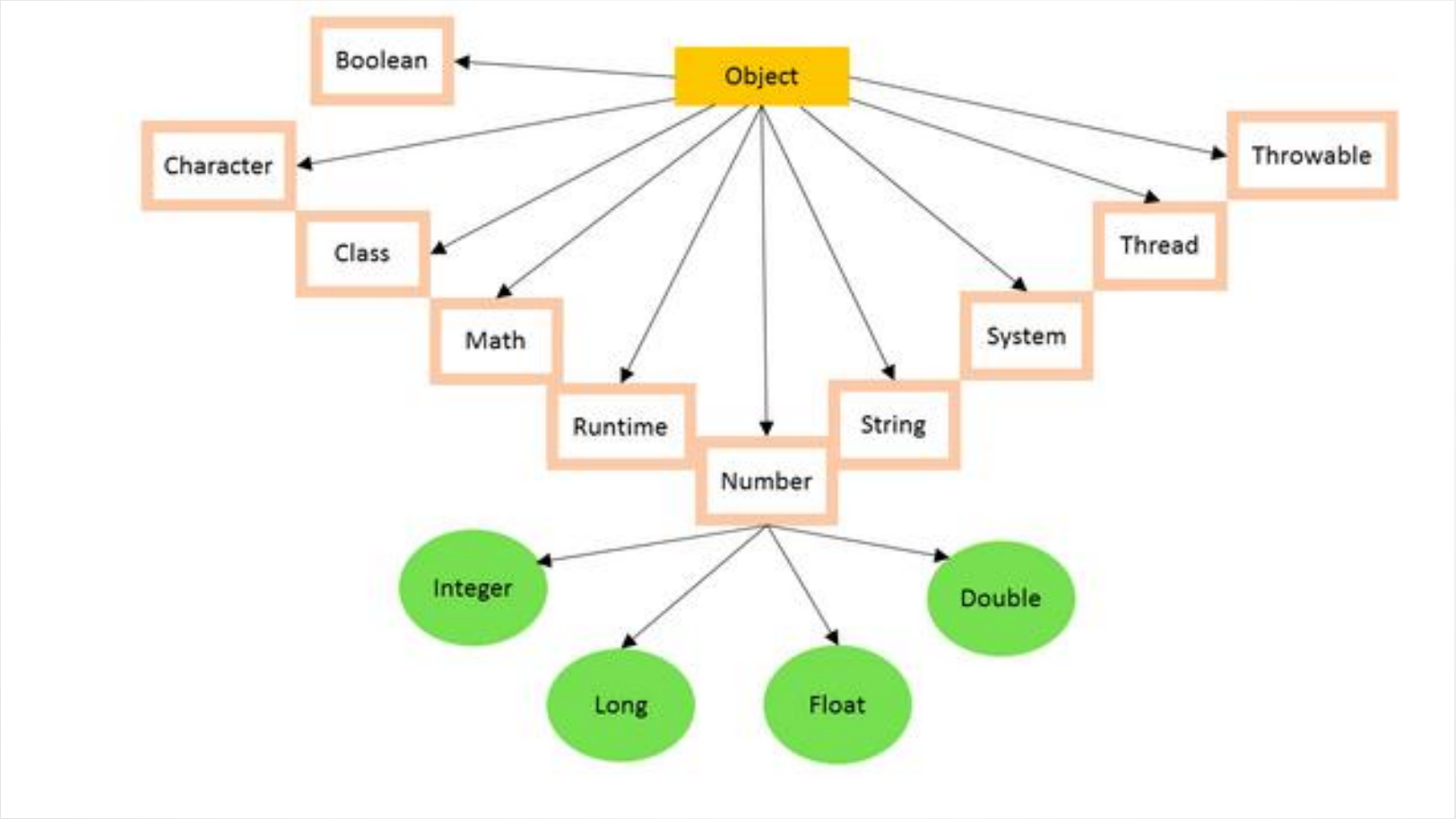
- The public access modifier is accessible everywhere.
- It has the widest scope among all other modifiers.

Access modifier	Within class	Within package	Outside package within subclass	Outside package
private	YES	NO	NO	NO
default	YES	YES	NO	NO
protected	YES	YES	YES	NO
public	YES	YES	YES	YES

Java.lang package

- Java.lang is imported by default in all the classes that we create.
- There is no need to explicitly import the lang package.
- It contains classes that form the basic building blocks of Java.
- There are 37 classes in java.lang package.
- Some of them are as follows:
  - Boolean
  - Byte
  - Double
  - Float
  - Integer
  - Long
  - Object
  - Short

Java.lang.Object class



Java.lang.Object class

- The Object class is the parent class of all the classes in java by default.
- In other words, it is the topmost class of java.
- The Object class is beneficial if you want to refer any object whose type you don't know.
- There is no need to explicitly inherit the Object class.

Java.lang.Object class

Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class<?> getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

Java Wrapper Classes

- Wrapper class in java provides the mechanism to convert primitive into object and object into primitive.
- For each primitive type, there is a corresponding wrapper class designed.
- An instance of wrapper contains or wraps a primitive value of the corresponding type.
- Wrappers allow for situations where primitives cannot be used but their corresponding objects are required.
- The eight classes of java.lang package are known as wrapper classes in java.
- The list of eight wrapper classes are given below:

Java Wrapper Classes

Wrapper Classes for Primitive Data Types	
Primitive Data Types	**Wrapper Classes**
int	Integer
short	Short
long	Long
byte	Byte
float	Float
double	Double
char	Character
boolean	Boolean

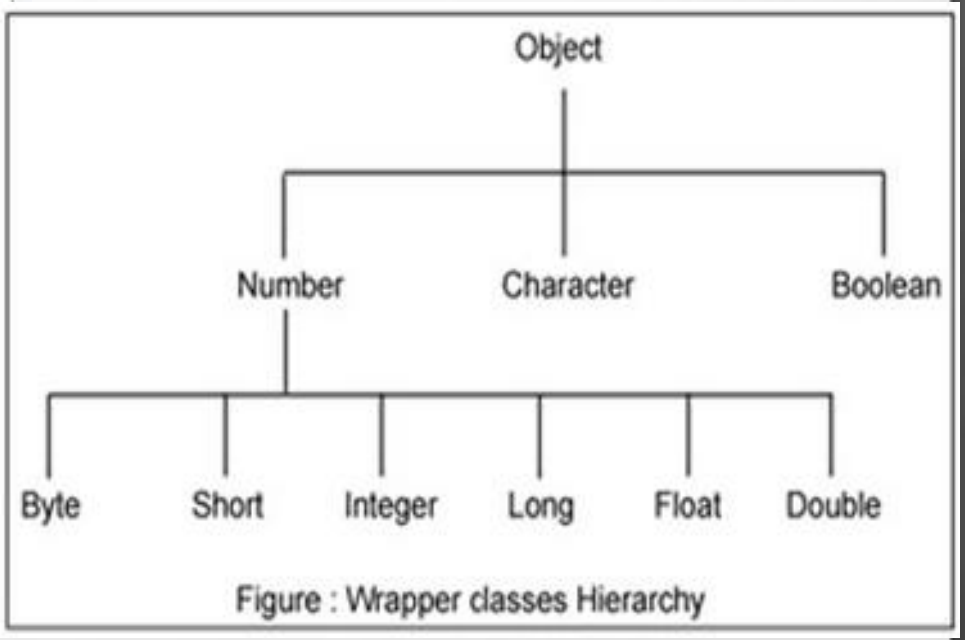


Figure : Wrapper classes Hierarchy

Java Wrapper Classes

- **Features of Wrapper Classes:**
  - All the wrapper classes except Character and Float have constructors. - one that takes the primitive value and another that takes the String representation of the value. Character has one constructor and float has three.
  - Just like strings, wrapper objects are also immutable. i.e once a value is assigned it cannot be changed.

Wrapper classes

- The wrapper classes have a number of static methods for handling and manipulating primitive data types and objects.
- **Constructors:** converting primitive types to wrapper classes

  Integer i = new Integer (10);
- **Methods:** converting wrapper objects to primitives

  all numeric classes have methods to convert a numeric wrapper class to their respective primitive type.

  byteValue(), intValue(), floatValue(), doubleValue(), longValue(), shortValue()

  int v = a.intValue();

Wrapper classes

- **Converting Primitives to String object:**

  the method toString() is used to convert primitive number data types to String

  String xyz = Integer.toString (v) //converting primitive integer to String

  String xyz = Float.toString (x)

- **Converting Back from String Object to Primitives:**

  the six parser methods are parseInt, parseDouble, parseFloat, parseLong, parseByte and parseShort.

  int v = Integer.parseInt (xyz);

Wrapper classes

- Wrapper classes are mainly used to wrap the primitive content into an object.
- This operation of wrapping primitive content into an object is called **boxing.**
- The reverse process i.e unwrapping the object into corresponding primitive data is called **Unboxing.**
- From JDK 1.5 onwards, **Auto-Boxing** is introduced. According to this feature, you need not to explicitly wrap the primitive content into an object. Just assign primitive data to corresponding wrapper class reference variable, java automatically wraps primitive data into corresponding wrapper object.
- From JDK 1.5 onwards, **Auto-Unboxing** is introduced. According to this feature, you need not to call method of wrapper class to unbox the wrapper object. Java implicitly converts wrapper object to corresponding primitive data if you assign wrapper object to primitive type variable.