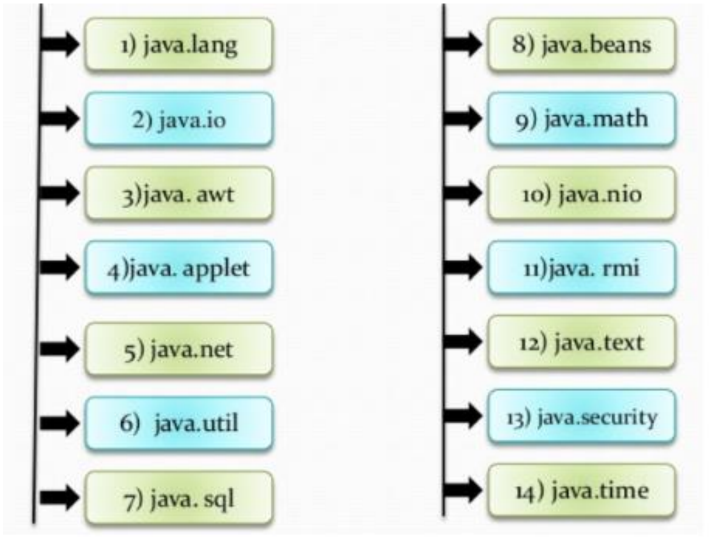
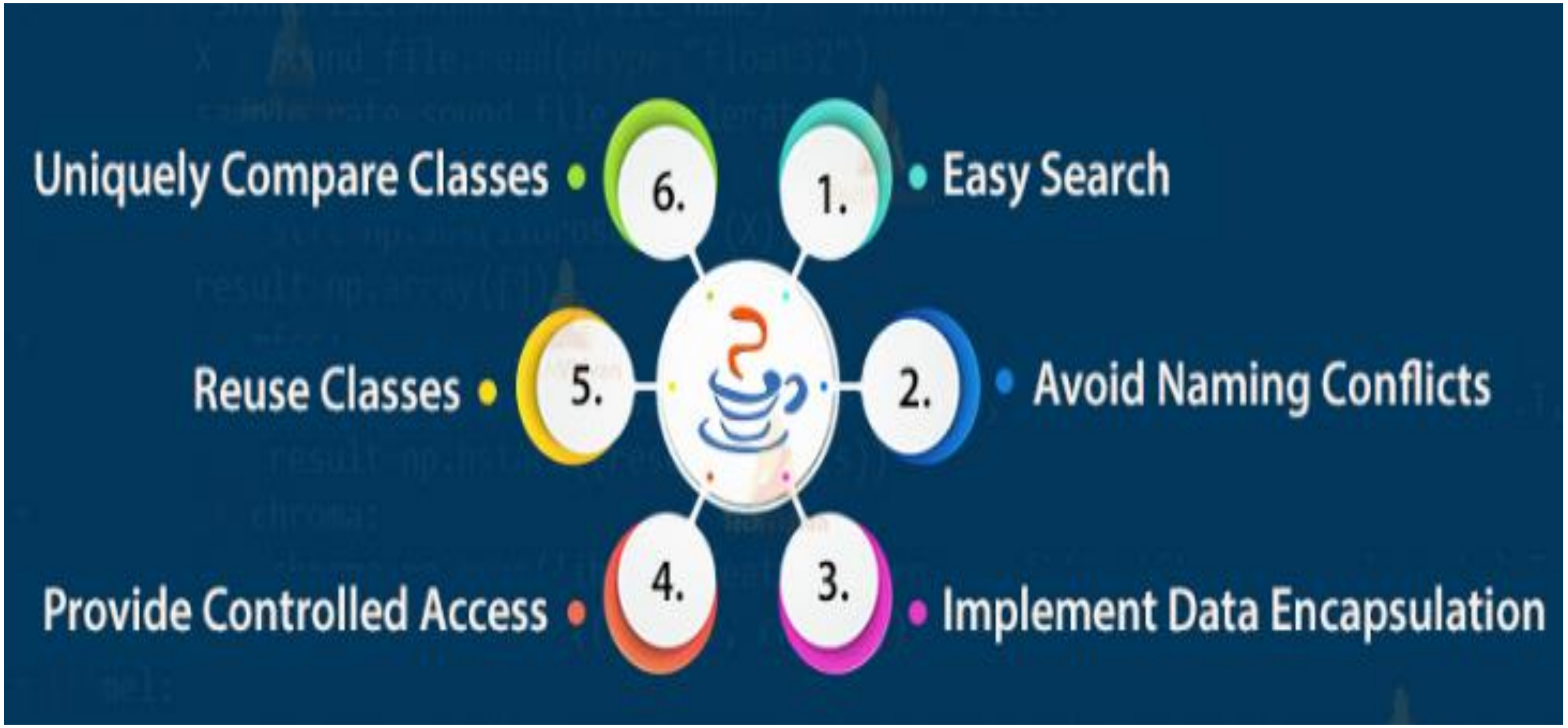
Java Packages

- A java package is a group of similar types of classes, interfaces and sub-packages.
- Types of packages :
  1. **built-in package – System defined packages**
  2. **user-defined package**
- Examples of built-in packages :
  - **java.lang**
  - **java. awt**
  - **Java.awt.event**
  - **java.io**
  - **java.util**
  - **java.applet**



fppt.com

Advantage of Java Package



Static Import

- The static import facilitate the Java programmer to access any static member of a class directly.
- **Advantage of static import**
  - Less coding is required if you have access any static member of a class often.
- Example:

```
import static java.lang.System.*;
class Static_Ex
{
    public static void main(String args[ ])
    {
      out.println("Hello");//Now no need of System.out
      out.println("Java");
    }
}
```

java.lang

- **java.lang is automatically imported into all programs**
- **java.lang includes the following classes:**

Boolean	**Long**	ThreadLocal
Byte	Math	StrictMath
Character	Number	**String**
Class	**Object**	**StringBuffer**
ClassLoader	Package	**System**
Compiler	Process	**Thread**
Double	Runtime	ThreadGroup
InheritableThreadLocal	RuntimePermission	StackTraceElement
Float	SecurityManager	Throwable
Integer	**Short**	**Void**

String

- String is a sequence of characters.
- In java, objects of String are immutable which means a constant and cannot be changed once created.
- Non - Primitive data types
- System defined class

    Ex. "AAA","A123","234","Java"

- Ways to create string :
    - String Literals

        String = "Atmiya";

    - Using new keyword

        String s = new String("Atmiya");

Constructors:

- **String( ) :**
    - Constructs a new empty String.

        Ex. String ct=new String()
- **String(String):**
    - Constructs a new String that is a copy of the specified String.

        Ex. String ct=new String("Rajkot")

Constructors…

- **String(char[ ]) :**
  - Constructs a new String whose initial value is the specified array of characters.

    Ex.   char[ ] c={‘r’, ’a’, ’j’, ’k’, ’o’, ‘t’};

    String ct=new String(c);

- **String(char[ ], int, int) :**
  - Constructs a new String whose initial value is the specified sub array of characters.

    Ex. char[] c={‘r’, ’a’, ’j’, ’k’, ’o’, ‘t’};

    String ct=new String(c,0,2); //raj

Constructors…

- **String(byte[]) :**
  - Constructs a new String whose initial value is the specified array of bytes.

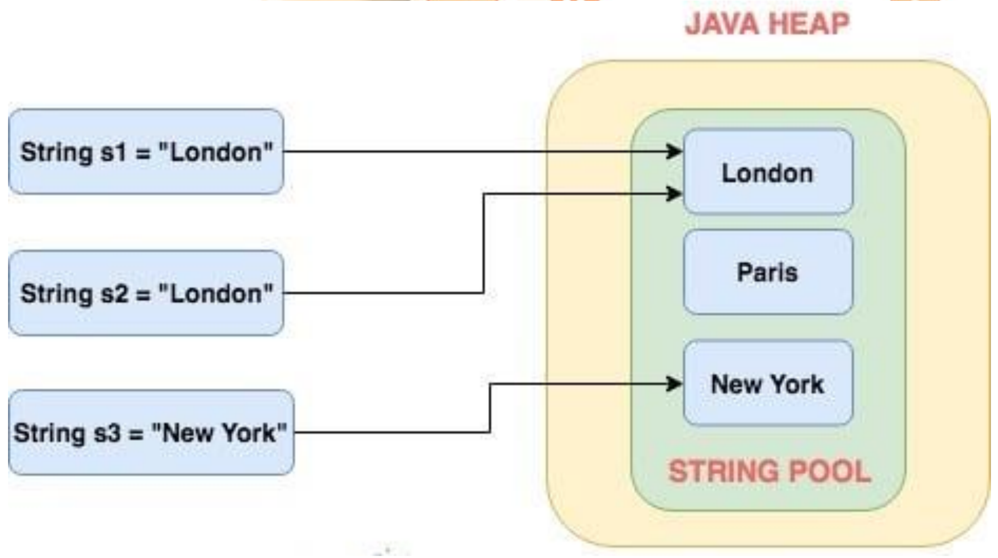
    Ex. byte[] b={65,66,67,68,69};

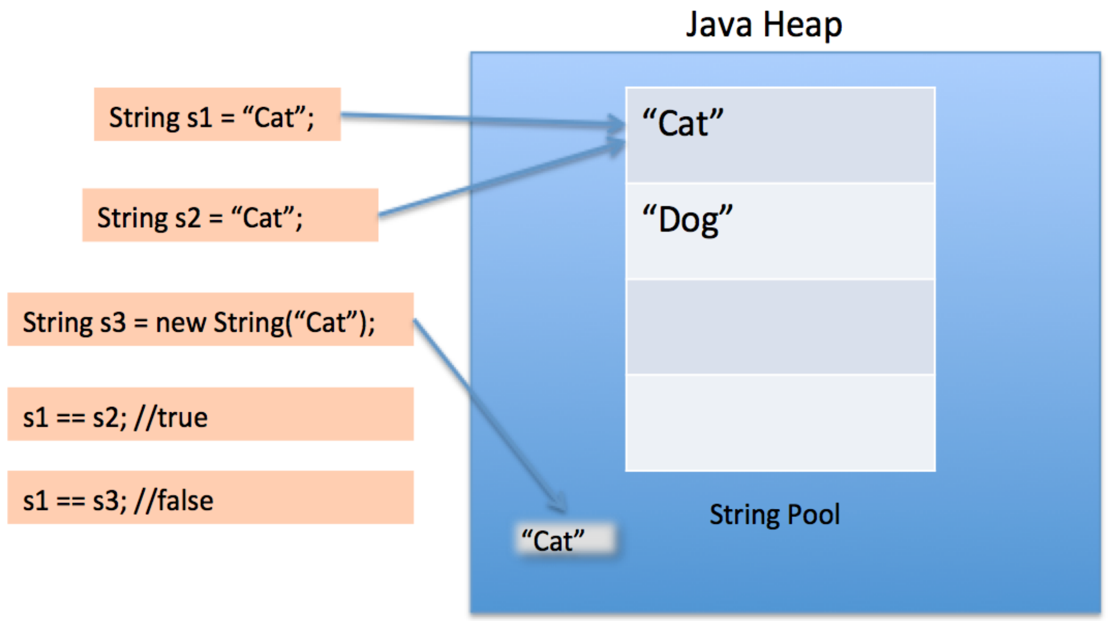
    String str=new String(b); //ABCDE

- **String(byte[], int, int) :**
  - Constructs a new String whose initial value is the specified sub array of bytes.

    Ex. byte[] b={65,66,67,68,69};

    String str=new String(b,2,3); //CD


JAVA HEAP
String s1 = "London"
String s2 = "London"
String s3 = "New York"
London
Paris
New York
STRING POOL


Java Heap
String s1 = “Cat”;
String s2 = “Cat”;
String s3 = new String(“Cat”);
s1 == s2; //true
s1 == s3; //false
“Cat”
“Dog”
String Pool
“Cat”

String Methods…

- **int length( ) :**
  - Returns the length of this string

    Ex. String str="Rajkot"

    int i=str.length( ) //6

- **String concat(String):**
  - Concatenates the specified string to the end of this String.
  - Returns the concatenated String.

    Ex. String str="Rajkot"

    str.concat(" Gujarat") //Rajkot Gujarat

- **char charAt(int) :**
  - Returns the character at the specified index.
  - e.g. String str=new String("atmiya")
    - char c= str.charAt(3); // i

Methods…

- **int compareTo(String):**
  - Compares this String to another specified String if match then return zero(0) otherwise not zero(0).

    Ex. String str = new String(“atmiya”);

    int ans = str.compareTo(“atmiya”) //0

- **int compareToIgnoreCase(String):**
  - Compares two strings , ignoring case differences.

    Ex. String str=new String(“Atmiya”)

    int ans = str.compareToIgnoreCase(“atmiya”) //0

- **byte[] getBytes( ) :**
  - Copies characters from this String into the specified byte array.

    Ex. byte b[]=new byte[10];

    String s=“Rajkot”

    b = s.getBytes( );

Methods…

- **char[] toCharArray() :**
  - Converts this String to a character array.

```
String s = “Atmiya”;
char c[]=new char[10];
 c = s.toCharArray( );
```

- **String toUpperCase() :**
  - Converts all of the characters in this String to upper case.

```
String s1,s2;
s1=“atmiya”;
s2=s1.toUpperCase( ) //ATMIYA
```

- **String toLowerCase( ) :**
  - Converts all of the characters in this String to lower case using.

```
String s1,s2;
s1=“ATMIYA”;
s2=s1.toLowerCase() //atmiya
```

Methods…

- **String substring(int, int) :**
  - Returns the substring of a String.

```
String s1,s2;
s1="ATMIYA";
s2=s1.substring(3,5) //IY
```

- **String replace(char, char) :**
  - Converts this String by replacing all occurrences of oldChar with newChar.

```
String s1,s2;
s1="atmiya";
s2=s1.replace('a','A') //AtmiyA
```

Methods…

- **int indexOf(String) :**
  - Returns the index within this String of the first occurrence of the specified substring.

- **int indexOf(String, int) :**
  - Returns the index within this String of the first occurrence of the specified substring.

    Ex. String s1="atmiya"

    int index=s1.indexOf("a");// 0

    int index=s1.indexOf("a",2);// 5

- **String[] split(String regex,int limit) :**
  - Splits the string

    String str="This is split function demo";

    String s1[]=str.split(" ",0); //s1[0]=This, s1[1]=is

String comparison

- **boolean equals(String) :**
  - checks whether two strings are equals or not.

    Ex. String str1 = “Hello”;

    String str2 = “hello”;

    str1.equals(str2) // returns false
- **boolean equalsIgnoreCase(String) :**
  - Ignore the case

    Ex. String str1 = “Hello”;

    String str2 = “hello”;

    str1.equalsIgnoreCase(str2) // returns true
- **boolean startsWith(String sub) :**
  - check whether string begin with specified sub string or not.

    Ex. String s = “Atmiya”;

    s.startsWith(“At”); // returns true
- **boolean endsWith(String sub) :**
  - check whether string end with specified sub string or not.

StringBuffer class

- Peer class of String class
- String represents **fixed length and immutable** character sequence.
- StringBuffer allows **growable and writable character** sequence.
- Characters in StringBuffer can be inserted / appended / added /deleted any where and the size of the StringBuffer will **automatically grow/shrink**.
  - StringBuffer str=new StringBuffer("VSC");

StringBuffer Constructors

- **StringBuffer( ) :**
  - Reserves room for 16 characters
- **StringBuffer(int size):**
  - Explicitly sets the size of buffer
- **StringBuffer(String str):**
  - Sets the initial content of the string Buffer and allocates room for 16 more characters

When no specific length is specified, StringBuffer allocates space for 16 more characters

Methods

Method	Description
StringBuffer append(String s)	is used to append the specified string with this string.
StringBuffer insert(int index, String s)	is used to insert the specified string with this string at the specified position.
StringBuffer replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
StringBuffer delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
StringBuffer reverse( )	is used to reverse the string.

Methods…

Method	Description
int capacity( )	return the current capacity.
char charAt(int index)	return the character at the specified position.
int length( )	return the length of the string
String substring(int beginIndex)	return the substring from the specified beginIndex.
String substring(int beginIndex, int endIndex)	return the substring from the specified begin and end Index.
int indexOf(String str)	returns the index within this string of the first occurrence of the specified substring.
void setCharAt(int index, char ch)	replace the given character at given place

Example

```
class A
{
    public static void main(String args[ ])
    {
        StringBuffer sb=new StringBuffer("Hello ");
        sb.append("Java");//original string is changed
        System.out.println(sb);//prints Hello Java
        sb.reverse();
        System.out.println(sb);//prints avaJ oellH
    }
}
```

Object class

- The **Object class** is the parent class of all the classes in Java by default.
- It is the topmost class of Java.
- All predefined classes and user defined classes are sub classed from Object class.
- Belongs into **java.lang** package.

Methods…

- public boolean **equals**(Object obj):-
  - compares the given object to this object.
- public String **toString() :-**
  - returns the string representation of this object.
- protected void **finalize():-**
  - is invoked by the garbage collector before object is being garbage collected.

Wrapper Classes

- The **wrapper class in Java** provides the mechanism *to* ***convert primitive into object and object into primitive***.
- Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.
- The automatic conversion of primitive into an object is known as **autoboxing** and vice-versa **unboxing.**

Datatype	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Hierarchy of wrapper classes

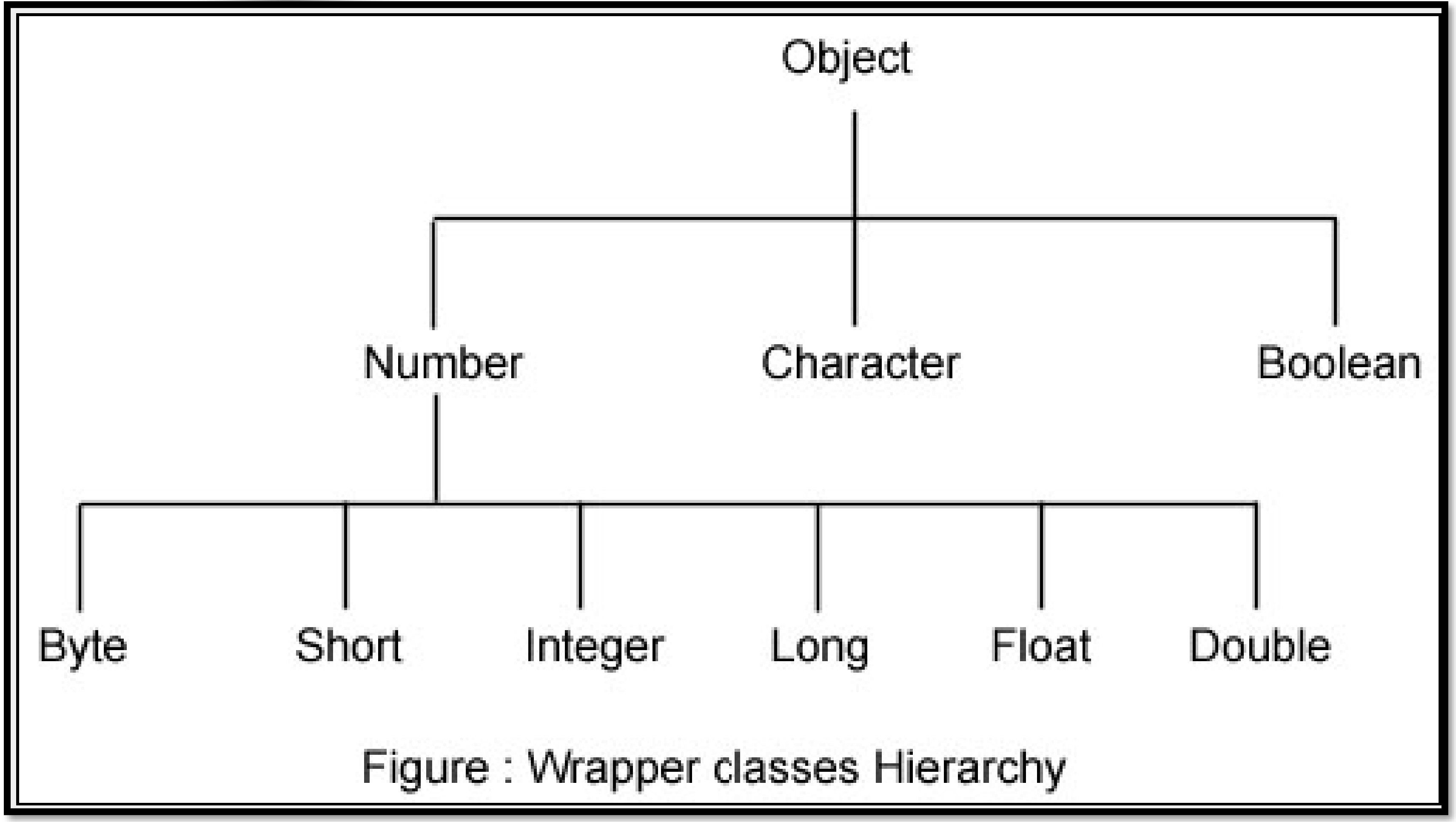


Figure : Wrapper classes Hierarchy

Convert Object num to primitive num

Method	Action
Integer val = new Interger(10); int i = **val.intValue( )**	Integer object to int
Float val = new Float(10.5); Float f = **val.floatValue( )**	Float object to float
Long val = new Long(100000); long l = **val.longValue( )**	Long object to long
Double val = new Double(10); double d = **val.doubleValue( )**	Double object to double

Converting numbers to String

- Using toString( ) any Number object can converted into String object.
- For Ex:
  - Integer val = new Integer(10);
  - String str = val.toString( ); //now str = “10”
- Likewise you can convert any number object to String object.

Convert String object to numeric object

- Using **valueOf( )** method of number class we can convert string object to particular number object.
- For Ex:
  - String str = “1234”;
  - Integer val = Integer.valueOf(str); // val = 1234
  - String s1= String.valueOf('a'); // s1="a"
  - String s2=String.valueOf(true); // s2="true"
- Likewise you can convert string object to any number object.

Convert numeric string to primitive data types

- Convert a String to any of the primitive data types, except character.
- All these methods are static.
- These methods have the format parse*() where * refers to any of the primitive data types except char.
- For ex:

```
String str = “34”;
int x = Integer.parseInt(str);    // x=34


String str = “34.45”;
double y = Double.parseDouble(str);   // y =34.45
```

USER DEFINED PACKAGE

Rules to create user defined package

- Package statement should be the first statement of any package program.
- Class name or interface name must be with modifier as public.
- Any package program can contain only one public class or only one public interface but it can contain any number of normal classes.
- Package program should not contain any main class (that means it should not contain any main())
- modifier of constructor of the class which is present in the package must be public.
- The modifier of method of class or interface which is present in the package must be public
- Every package program should be save either with public class name or public Interface name

Points to be remembered

- A package is always defined as a separate folder having the same name as the package name.
- Store all the classes in that package folder.
- All classes of the package which we wish to access outside the package must be declared public.
- All classes within the package must have the package statement as its first line.
- All classes of the package must be compiled before use.

How to access package from another package?

- Three ways to access the package from outside the package.

1. import package.*;
    - All the classes and interfaces of this package will be accessible but not subpackages.
2. import package.classname;
    - Only declared class of this package will be accessible.
3. fully qualified name
    - Only declared class of this package will be accessible.
    - No need to import.
    - But you need to use fully qualified name every time when you are accessing the class or interface.

How to access package from another package?

- For import the Scanner class from util package.

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

- For import all the classes from util package

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

Importing by using full qualified path name

```
//save by A.java
    package pack;
    public class A
{
     public void msg(){System.out.println("Hello");}
}
//save by B.java
    package mypack;
    class B
{
     public static void main(String args[]){
      pack.A obj = new pack.A();//using fully qualified name
     obj.msg();
 }
}
```

Example for User Defined Package

- The package keyword is used to create a package in java.

```
package mypack;
public class Simple_Pack
{
        public static void main(String args[ ])
        {
                System.out.println("Welcome to package");
        }
 }
```

- Compile: **javac –d . Simple_Pack.java**
  (Here dot represents current directory)
- Run : **java mypack.Simple_Pack**

Output:

Another Example:

Access package from another package

```
package pack;
public class A
{   public void msg()
    {       System.out.println("Hello");            }
}
```

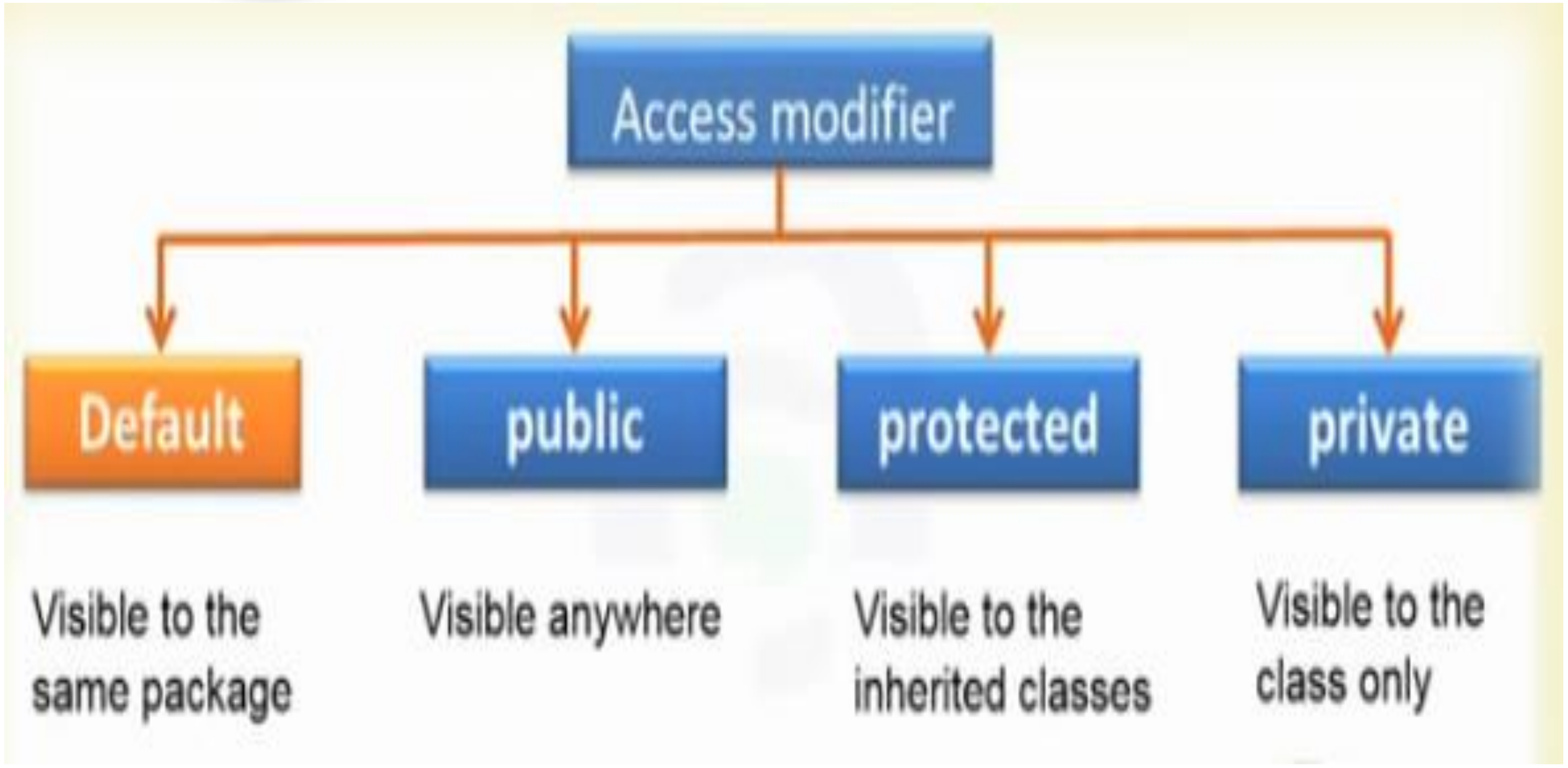
- ======================

```
package mypack;
import pack.*;
class B
{   public static void main(String args[ ])
    {    A obj = new A( );
           obj.msg();
    }
}
```

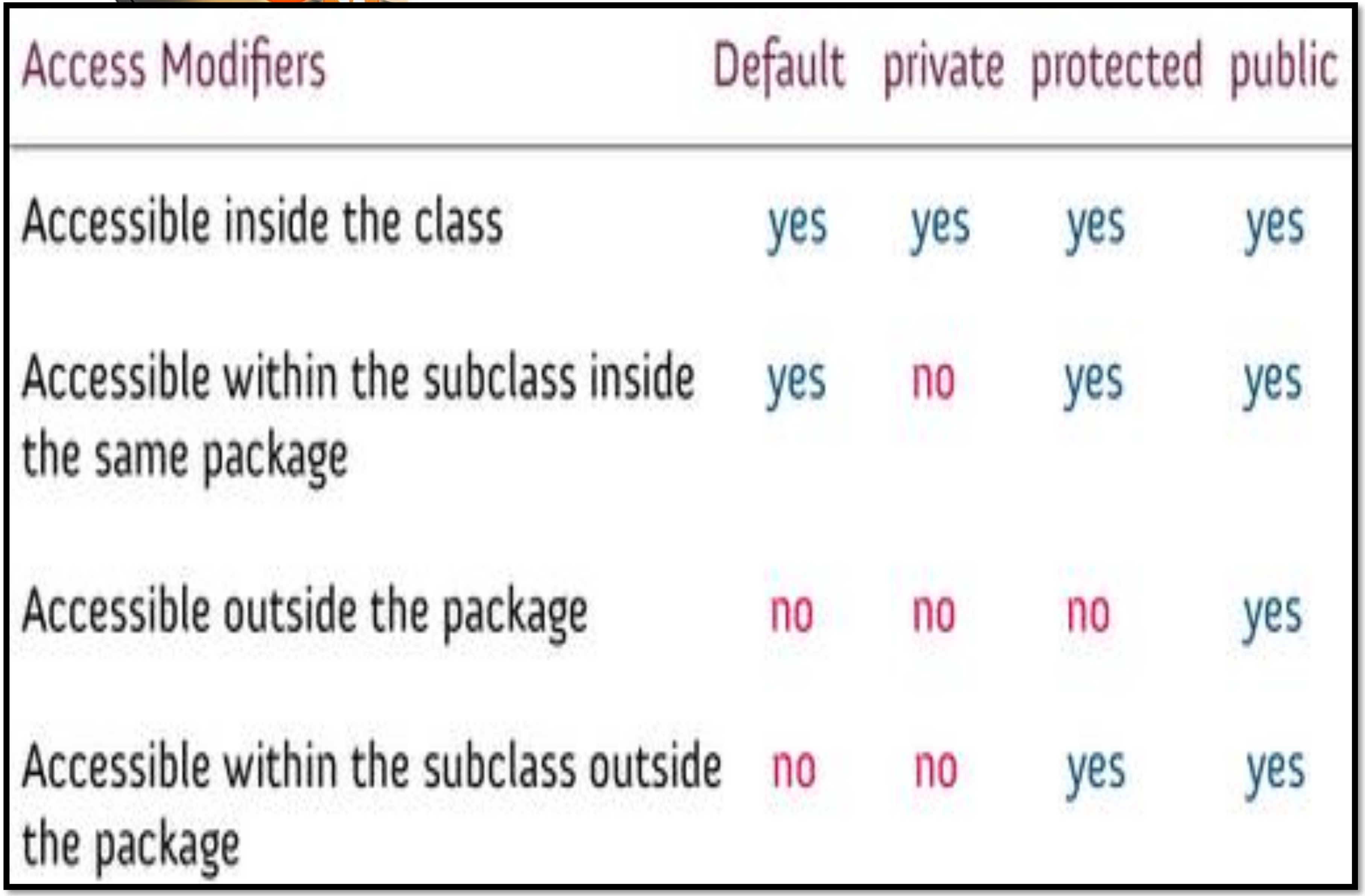
- Example A
- Example B

Access Specifiers

- Java Access Specifiers (Visibility Specifiers) regulate access to classes, attributes and methods in Java.
- It determine whether attributes or methods in a class, can be used or invoked by another method in another class or sub-class.
- Access Specifiers can be used to restrict access.
- Access Specifiers are part of object-oriented programming.
- **Types Of Access Specifiers :**
  1. public
  2. private
  3. protected
  4. default(no specifier)


Access modifier
Default
public
protected
private
Visible to the same package
Visible anywhere
Visible to the inherited classes
Visible to the class only

Access Specifiers…



Access Modifiers	Default	private	protected	public
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes