Bharathidasan University

Centre for Differently Abled Persons
Tiruchirappalli - 620024.

Programme Name	: Bachelor of Computer Applications
Course Code	: 23UCACC04
Course Title	: Programming in Java
Semester	: IV
Unit	: Unit II
Compiled by	: Dr. M. Prabavathy Associate Professor Ms. M. Hemalatha Guest Faculty

Java™

Classes & Objects

CLASSES

- Class is the **blueprint of an Object**.
- It is the basic building block of an object-oriented language.

A class in Java can contain:

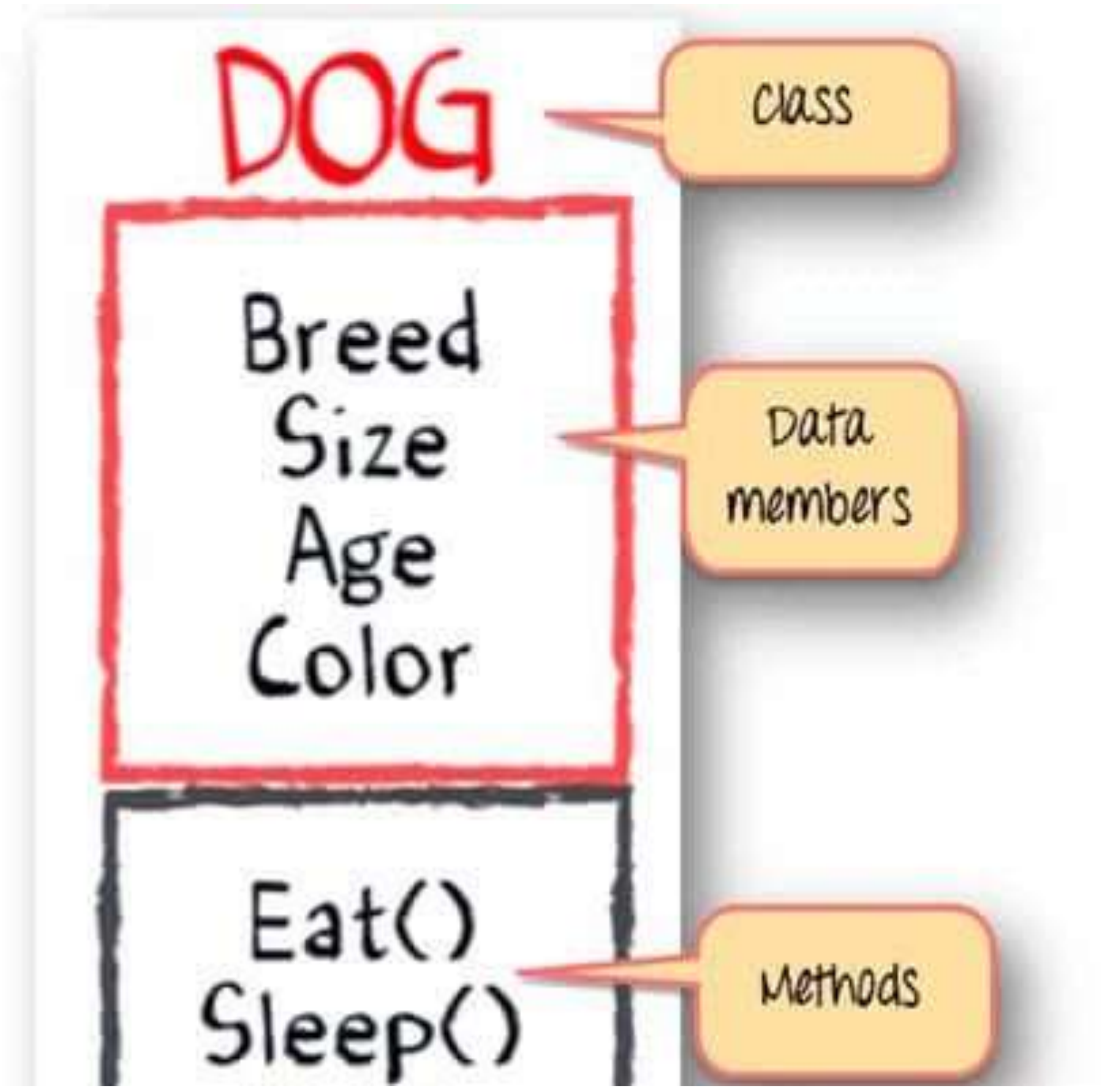
- Fields / Data Members
- Methods
- Constructors
- Blocks
- Nested class and interface.

Syntax:

```
class<class_name>
{
        Field / Data Members;
        Methods;
}
```

Example:

```
class Dog
{
	String breed;
	int size;
	int age;
	String color;
void eat()
{
	System.out.println(“ The Dog is Eating”);
}
void sleep()
{
	System.out.println(“ The Dog is Sleeping”);
}
}
```


DOG
Class
Breed
Size
Age
Color
Data members
Eat()
Sleep()
Methods

OBJECTS

- An object is called an **instance of a class.**
- **New** keyword is used to create Object.
- Object **allocates memory** when it is created

Syntax:

ClassName ObjectName = New ClassName();

Creating an Object

There are three steps:

1. **Declaration** – Declare variable name with an object type.

2. **Instantiation** – The 'new' keyword is used to create the object.

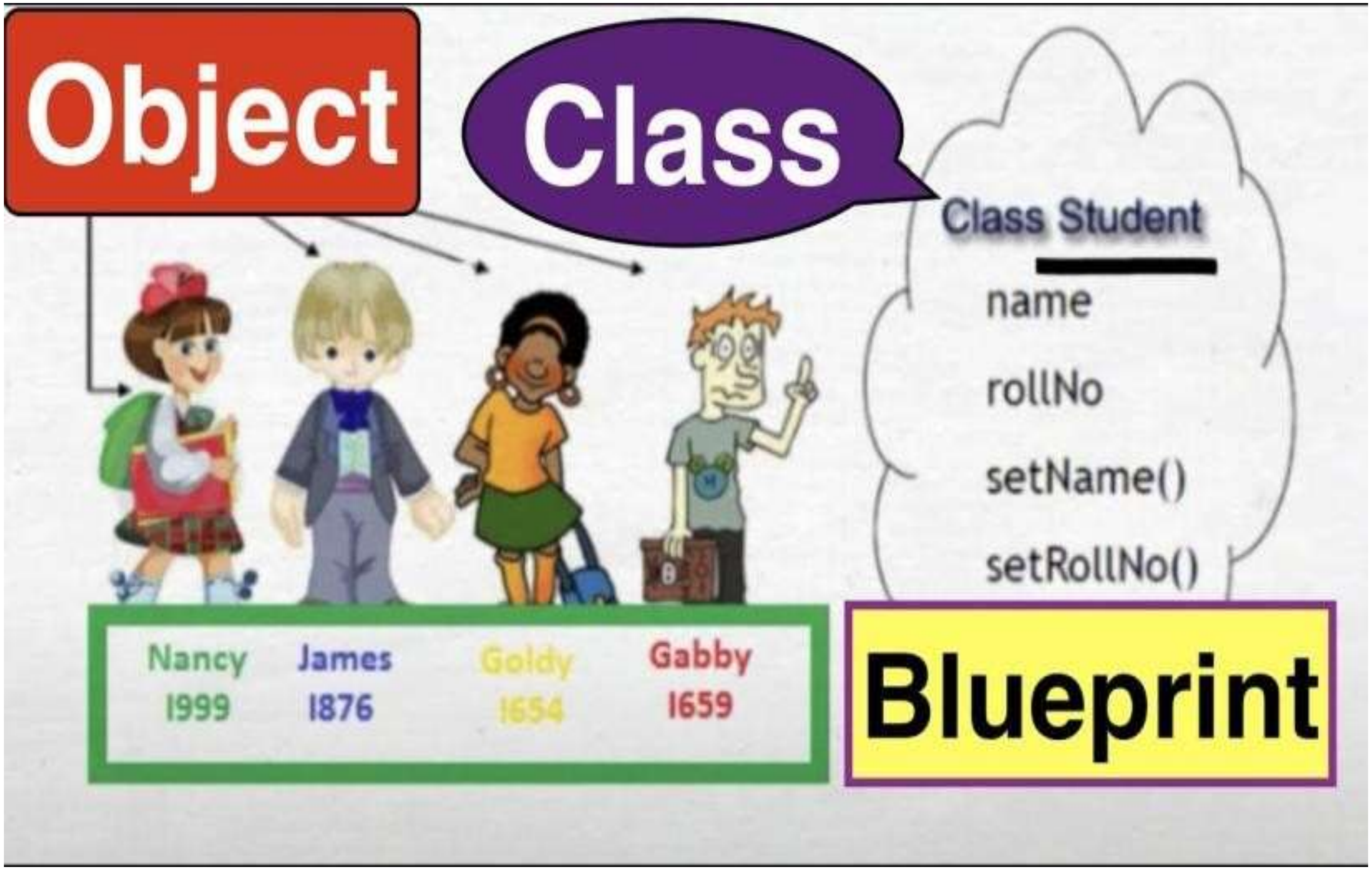
3. **Initialization** – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example:

```
class Student
{
    int rollno;
    String name;
}
class TestStudent2
{
public static void main(String args[])
{
    Student s1=new Student();
    s1.rollno=101;
    s1.name="CDAP";
    System.out.println(s1.id+" "+s1.name);
}
}
```

OUTPUT

```
                101       CDAP
```


Object
Class
Class Student
name
rollNo
setName()
setRollNo()
Nancy 1999
James 1876
Goldy 1654
Gabby 1659
Blueprint

CONSTRUCTOR

CONSTRUCTOR

- 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.

Rules for creating Java constructor

1. Constructor name must be the same as its class name

2. A Constructor must have no explicit return type

3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

- Default constructor
- Parameterized constructor

1. Default Constructor

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

Syntax

```
<class_name>( )
{ code; }
```

```
classTest1
{
Test1( ) ⟶ Default Constructor
   {
        System.out.println("Have a Good Day");
   }

public static void main(String args[])
{
Test1 b=new Test1();
}
}
```

Output:

```
        Have a Good Day
```

2. Parameterized Constructor

A constructor which has a **specific number of parameters**

Example

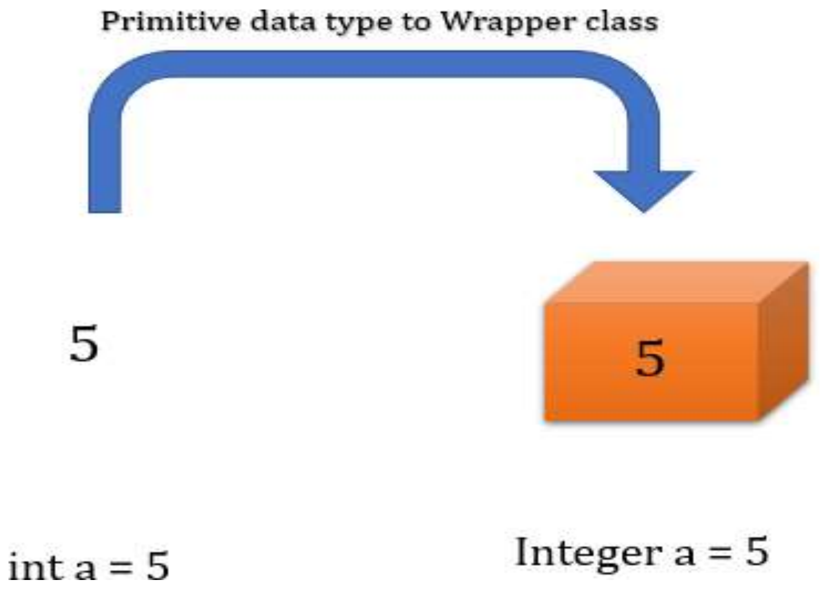
```
class Student4
{
        int id;
        String name;
Student4(int i, String n) → Parameterized Constructor
{
        id = i;
        name = n;
}

void display()
{
        System.out.println(id+" "+name);
}
public static void main(String args[])
{
   Student4 s1 = new Student4(101,"Aarthi");
   Student4 s2 = new Student4(102,"Abinaya");
   s1.display();
   s2.display();
}
}
```

Output:

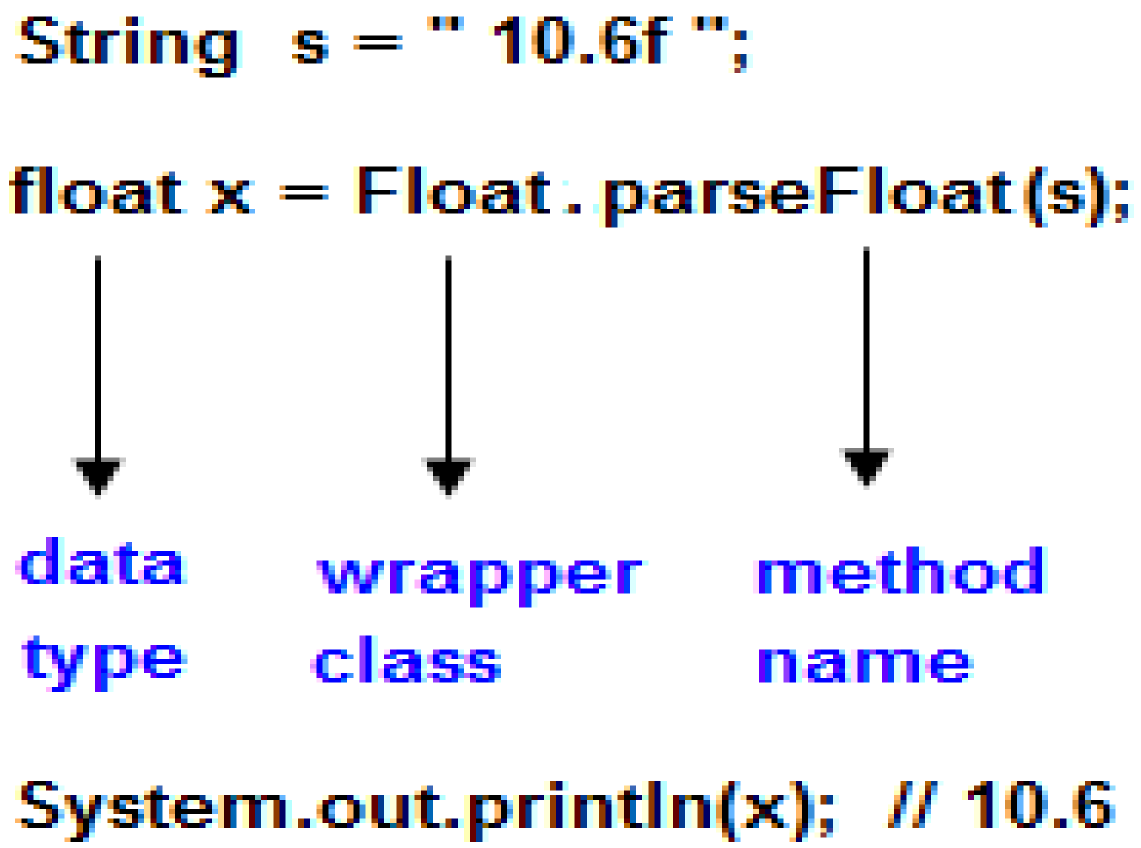
```
101    Aarthi
102    Abinaya
```

Wrapper classes

- The wrapper classes are used to convert **primitive types** (int , char , float, etc.) into **corresponding objects.**

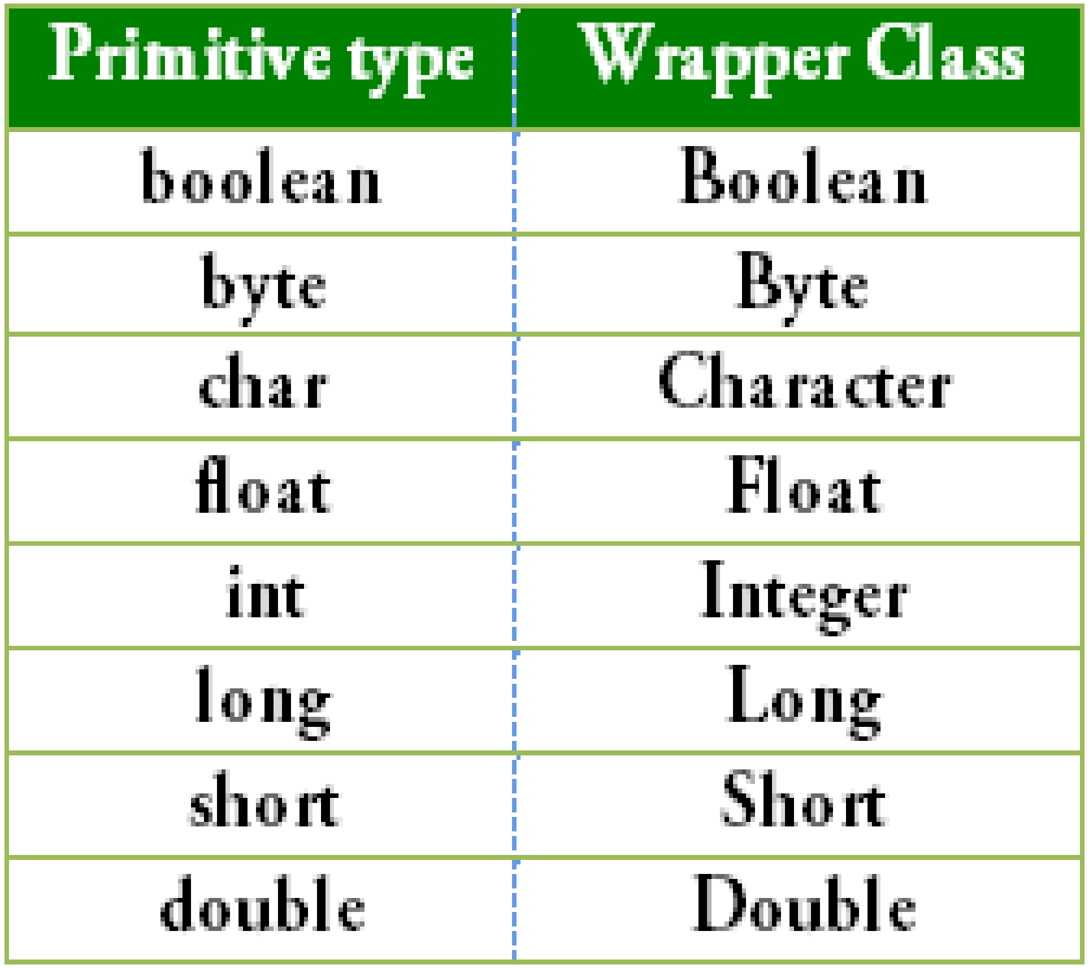


Example



```
String  s = " 10.6f ";

float x = Float.parseFloat(s);
```

```
System.out.println(x);  // 10.6
```



Primitive type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Autoboxing

The automatic conversion of **primitive data type into its corresponding wrapper class** is known as Autoboxing

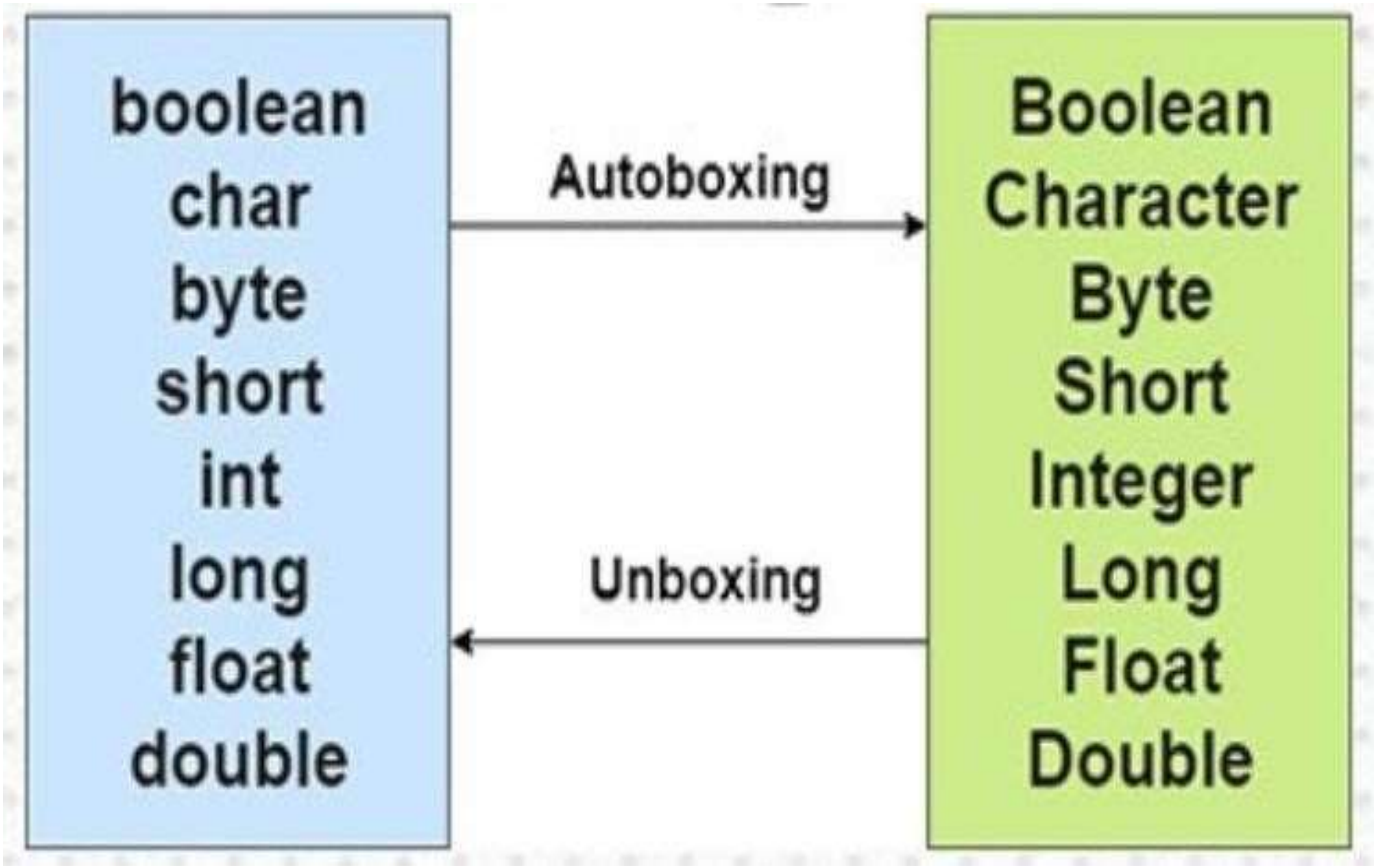
Examples:

- byte to Byte
- char to Character
- int to Integer

Unboxing

The automatic conversion of **wrapper type** into its **corresponding primitive type** is known as Unboxing.

It is the reverse process of Autoboxing.


boolean
char
byte
short
int
long
float
double
Autoboxing
Unboxing
Boolean
Character
Byte
Short
Integer
Long
Float
Double

INTERFACES

INTERFACES

- An interface is a **blueprint of a class.**

- It has static constants and abstract methods.

- There can be only **abstract methods** in the Java interface, not method body.

Declaring Interface

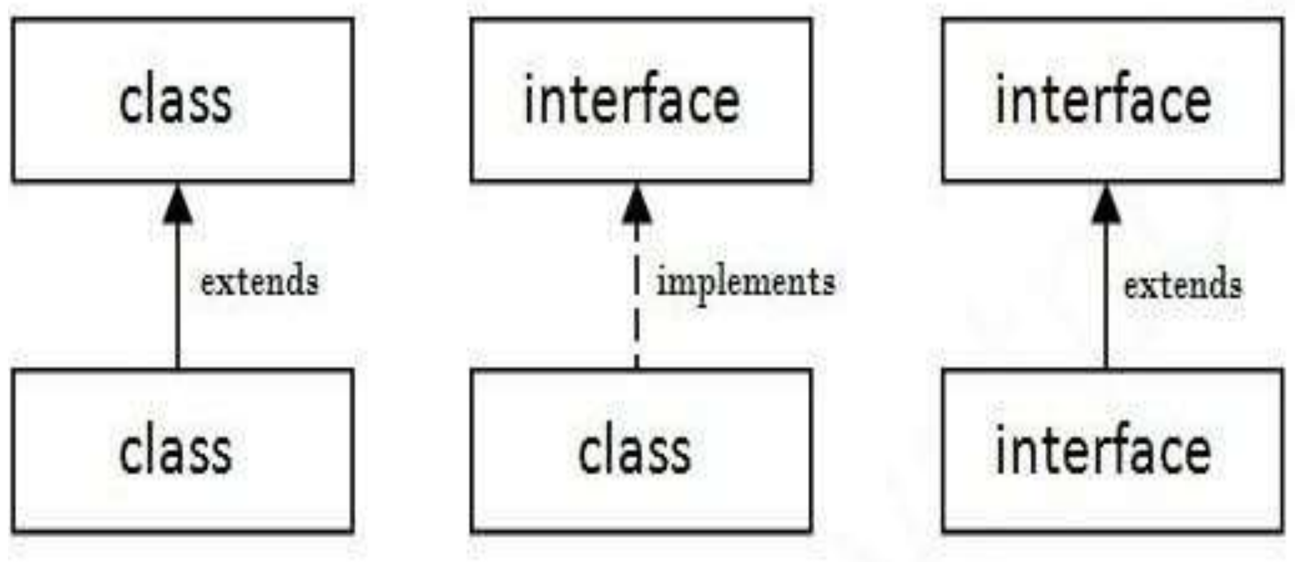
```
interface<interface_name>
{
      // declare constant fields
      // declare methods that abstract
      // by default.
}
```

Example

```
interface printable
{
        void print();
}
class A6 implements printable
{
public void print()
{
        System.out.println("Hello");
}
public static void main(String args[])
{
        A6 obj = new A6(); obj.print();
}
}
```

Relation between Class and Interface

A class extends another class, an interface extends another interface, but a class implements an interface.



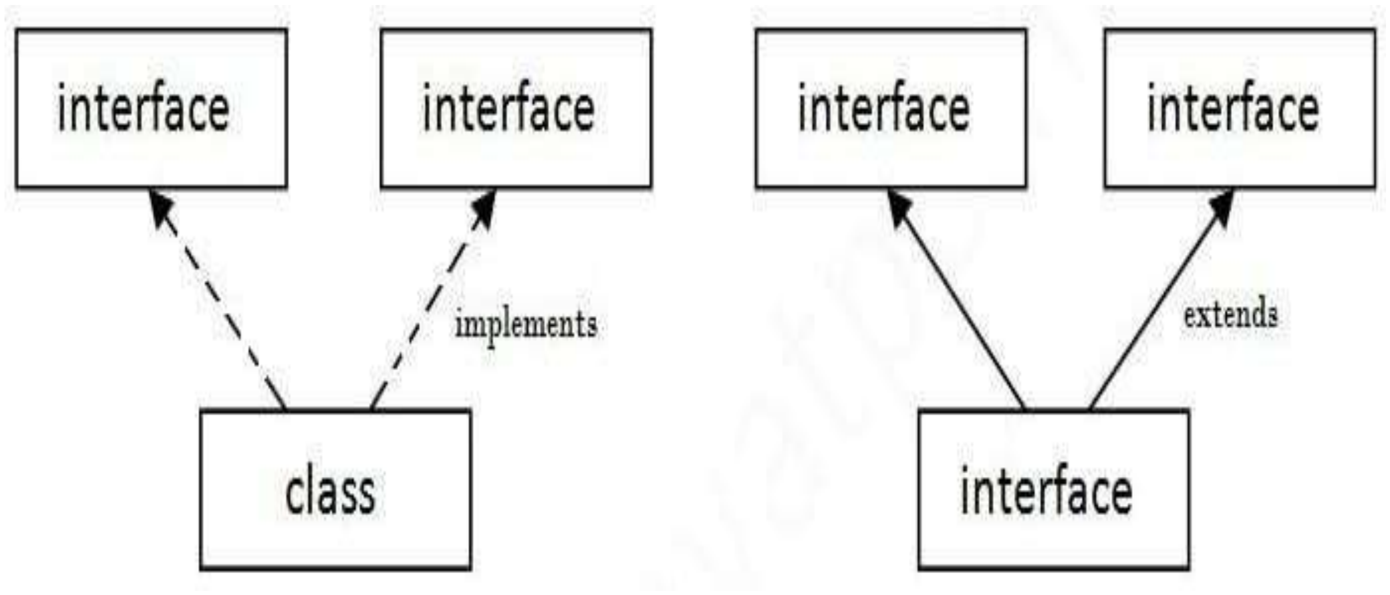
Multiple Inheritance

If a **class implements multiple interfaces**,

or

an interface extends multiple interfaces,

it is known as multiple inheritance.



Example:

```
interface Printable
{
      void print();
}
interface Showable
{
      void show();
}
```

```
class A7 implements Printable, Showable
{
public void print()
{
        System.out.println("Hello");
}
public void show()
{
        System.out.println("Welcome");
}
public static void main(String args[])
{
        A7 obj = new A7();
        obj.print();
        obj.show();
}
}
```

OUTPUT

```
                Hello
                Welcome
```

PACKAGES

Packages

- A java package is a **group of similar types of classes, interfaces and sub- packages**.

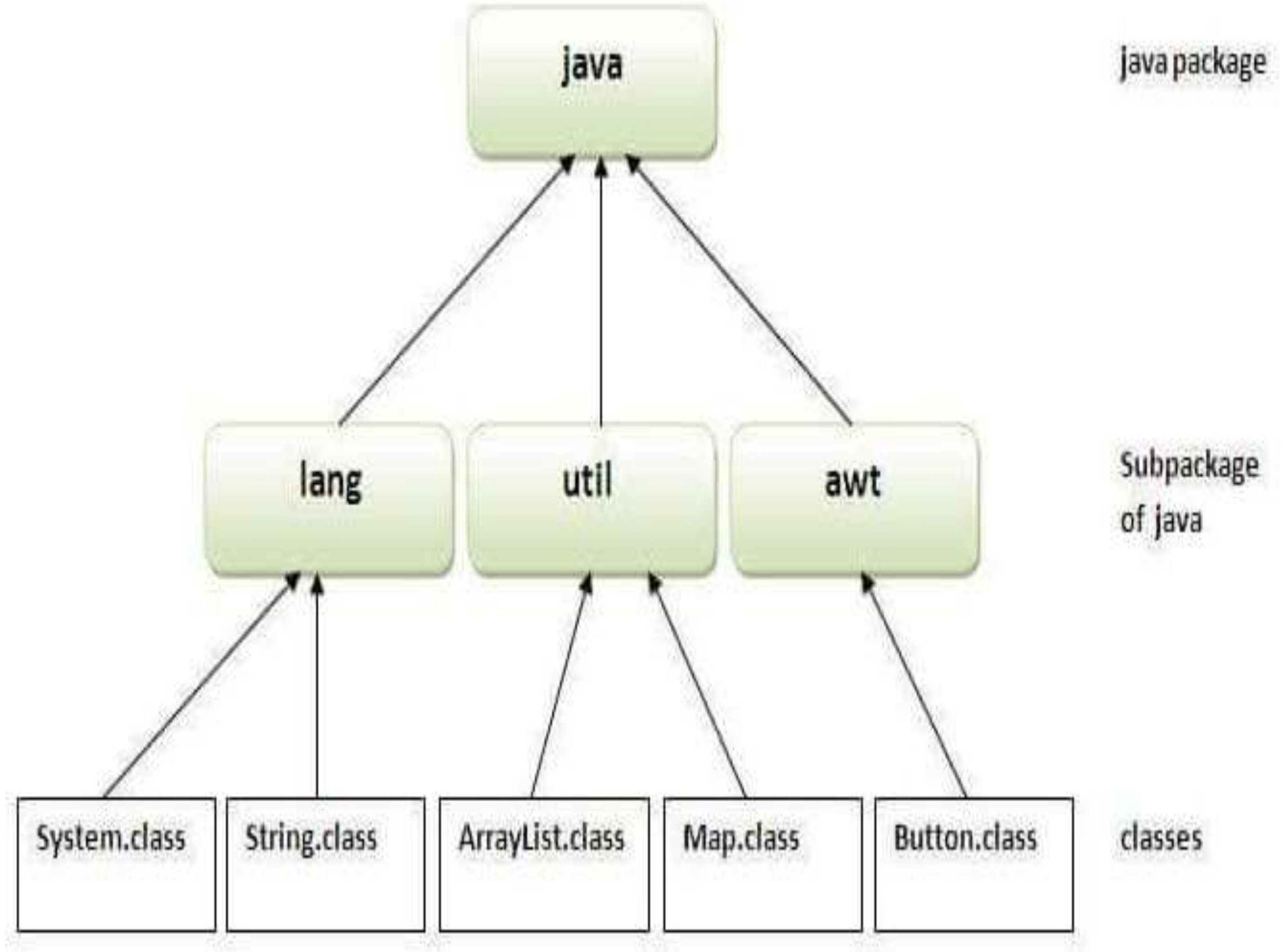
- Package in java can be categorized in two:

built-in package.

user-defined package.

- There are many built-in packages such as java,lang, awt, javax, swing, net, io, util, sql etc.

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


java
java package
lang
util
awt
Subpackage of java
System.class
String.class
ArrayList.class
Map.class
Button.class
classes

Built-in Packages

These packages consist of a large number of classes which are a part of Java API.

Some of the commonly used built-in packages are:

1. java.lang

Contains language **support classes**

Example: classes which defines primitive data types, math operations

2. java.io

Contains classed for supporting **input / output operations.**

3. java.util

Contains **utility classes** which implement data structures like **Linked List, Dictionary** and support **Date / Time operations**.

4. java.applet

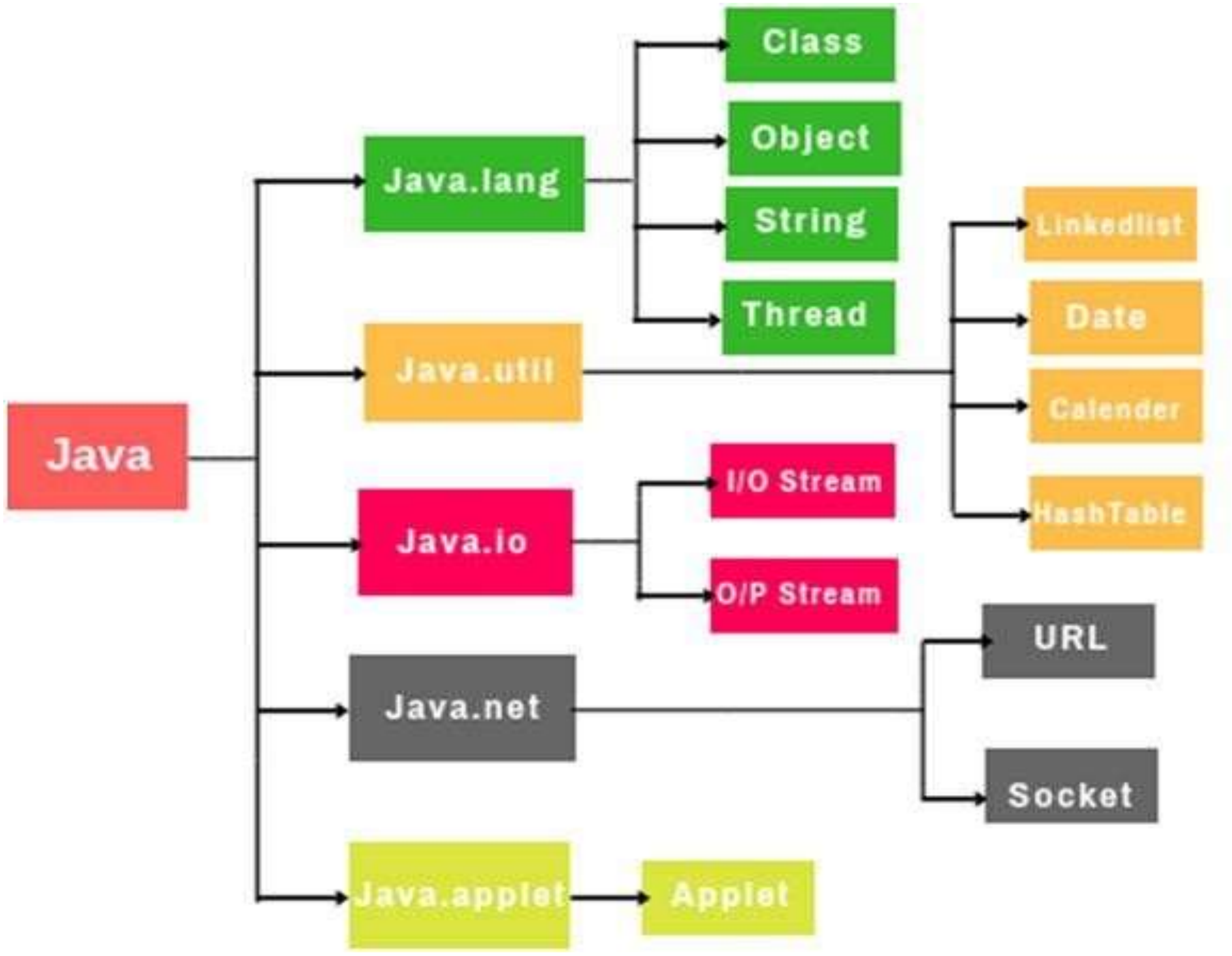
Contains classes for **creating Applets**.

5. Java.awt

Contain classes for implementing the components for **graphical user interfaces** (like button, menus etc).

6. java.net

Contain classes for supporting **networking operations.**


Java
Java.lang
Class
Object
String
Thread
Java.util
Linkedlist
Date
Calender
HashTable
Java.io
I/O Stream
O/P Stream
Java.net
URL
Socket
Java.applet
Applet

User Defined Package

To create your own package

Java uses a file system directory to store them

Just like folders on computer

Example

```
└── root
    └── mypack
        └── MyPackageClass.java
```

To create a package, use the package keyword

Sample Program for Package

// A.java

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

//B.java

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

Output:

```
WELCOME TO CDAP
```

How to Run Java Package

1. Compile a java program which is defined as Package **javac A.java**
2. Create a **folder** and name it as **pack(Package Name)**
3. **Copy** the source file **(A.java)** and class file **(A.class)**
4. **Paste** the copied files into **pack folder**
5. Now write the program for **class B**
6. **Compile** and **run** the program

```
javac B.java

java B
```

7. Thus the user defined package (pack) was imported and display the output as

WELCOME TO CDAP

THANK YOU