JAVA (Skill Based Lab)

MODULE 2: CLASSES, OBJECTS, PACKAGES, INPUT AND OUTPUT

Prof Indu Anoop

What's in this Module

01

1.1.CLASSES

1.2.OBJECTS

1.3 DATA MEMBERS AND MEMBER FUNCTIONS

02

2.1 CONSTRUCTORS

TYPES

2.2 STATIC MEMBERS AND FUNCTIONS

2.3 METHOD OVERLOADING

03

PACKAGES IN JAVA: TYPES, USER DEFINDED PACKAGES

04

INPUT/OUTPUT FUNCTIONS:SCANNER, BUFFERED READER

1.1 Class

A class is a blueprint from which individual objects are created. 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. We can create a class, using the keyword “class”

```
class Dog
{
   String breed;
   int age;
   String color;

   void barking() {
   }
   void hungry() {
   }
   void sleeping() {
   }
}
```

1.1 Class : Syntax

File Name: Simple.java

Class Name

Comment

Main method

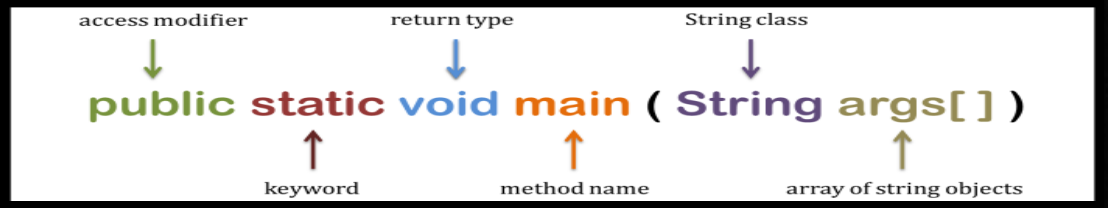
```
class Simple {

    // This is my first java program.
    public static void main(String args[])
     {
      System.out.println("Hello World!");
     }
}
```

access modifier/ specifier

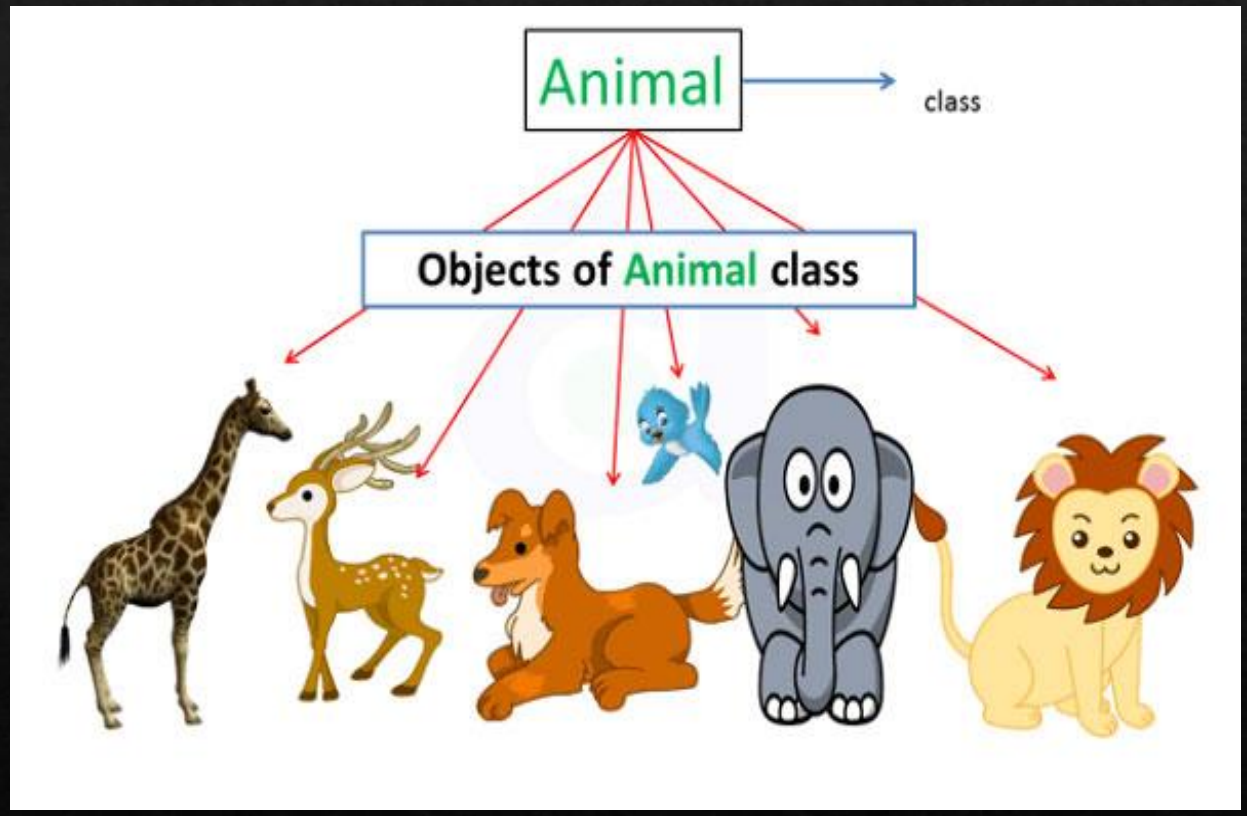
Method return type

Body of Main method



1.2 Object

- Object are defined as the instance(representation) of a class.
- Example table, chair, are all instance of the class Furniture.
- Object of a class will have same attribute and function(method) which are defined in that class.
- The only difference between objects of same class will be in values of attribute.
- Each Object Entity have unique identity and behavior



1.2 Object (cont'd)

An object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class

Declaration – A variable declaration with a variable name with an object type.

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

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

```
public class Puppy
{
  public Puppy(String name)
{
    // This constructor has one parameter, name.
    System.out.println("Passed Name is :" + name );
  }
  public static void main(String []args)
  {
    // Following statement would create an object myPuppy
    Puppy myPuppy = new Puppy( "tommy" );
  }
}
```

Output: Passed Name is :tommy

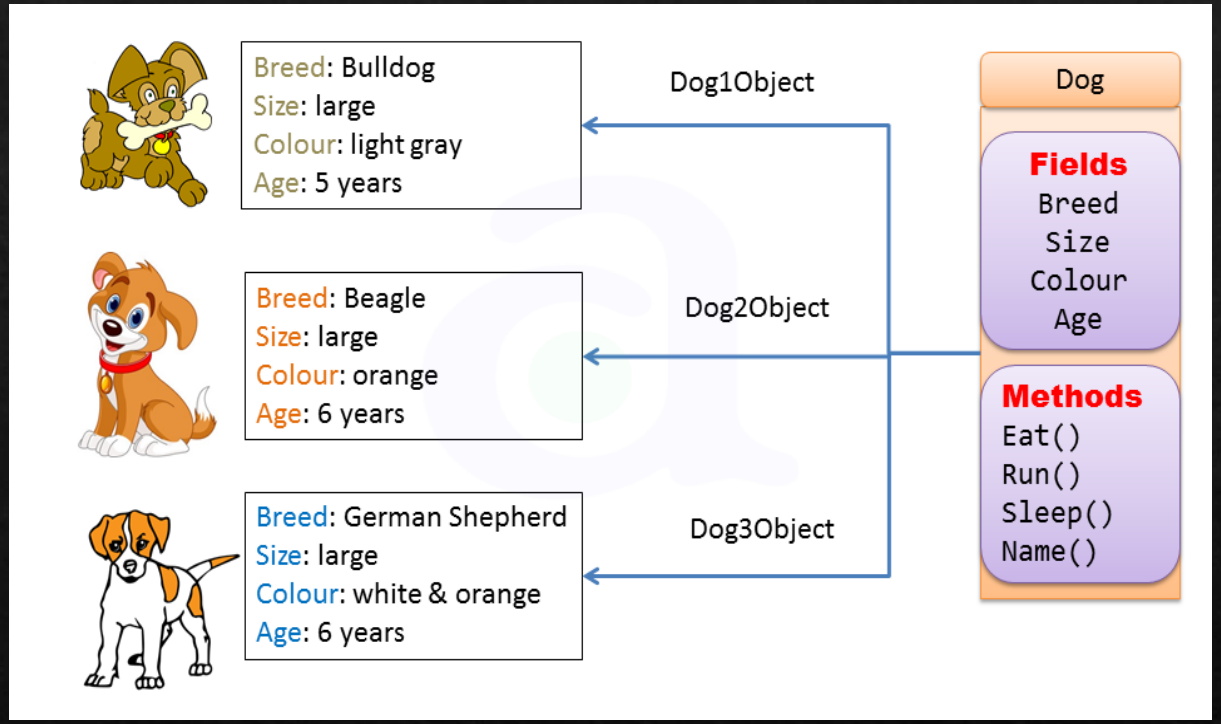
1.3 DATA MEMBER AND MEMBER FUNCTIONS

DATA MEMBERS: Variables declared within a class preceded by a data type which define the state of an object are called data members
MEMBER FUNCTIONS: functions which define the behavior of an object of that class are called member functions

```
class Dog{
    String breed;
    int size;
    String colour;
    int age;
    void eat(){
    }
    void run(){
    }
    void sleep(){
    }
    void name(){
    }
}
```

Data Members

Member Functions



2.1 Constructors: Introduction

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

Rules for creating Java 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

Eg:-

```
class Animal{
    public static void main(String[] args){
        Animal a = new Animal();
    }
}
```

Default constructor is called

2.1 Constructors: Types

Types of Java constructors: There are three types of constructors in Java:

- Default constructor (no arguments): The Java compiler provides a default constructor if you don't have any constructor in a class. The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.
- No arguments constructor (user-defined): Similar to the default constructor but is user defined with no arguments.
- Parameterized constructor (user-defined): A constructor which has a specific number of parameters is called a parameterized constructor. The parameterized constructor is used to provide different values to distinct objects.

Default constructor

```
class Test{
   int i;
   public static void main(String[] args){
      Test t = new Test();
      System.out.println(t.i);
   }
}
```

output will be → 0

No-args (user-defined)

```
class Test{
   int i;
   Test(){
      i = 6;
   }
   public static void main(String[] args){
      Test t = new Test();
      System.out.println(t.i);
   }
}
```

output will be → 6

Parameterized constructor

```
class Test{
   Test(String name)
   {
   System.out.println(name);
   }
   public static void main
   (String[] args){
      Test t = new Test("Indu");
   }
```

output will be → Indu

2.2 Static Methods and Members

In Java, static members are those which belongs to the class and you can access these members without instantiating the class. The static keyword can be used with methods, fields, classes (inner/nested), blocks.

We can create a static method by using the keyword static. Static methods can access only static fields, methods. To access static methods there is no need to instantiate the class, you can do it just using the class name

```
public class MyClass {
        public static void sample() {
                System.out.println("Hello");
        }
        public static void main(String args[]) {
                MyClass.sample();
        }
}
```

Output will be → Hello

Static Data Members: We can create a static field by using the keyword static. The static fields have the same value in all the instances of the class. These are created and initialized when the class is loaded for the first time. Just like static methods you can access static fields using the class name (without instantiation).

```
public class MyClass {
        public static int data = 20;
        public static void main(String args[]) {
                System.out.println(MyClass.data);
        }
}
```

Output will be → 20

2.3 Method Overloading (Static Polymorphism)

Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both. Overloading is related to compile-time (or static) polymorphism. Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.

- Two methods are said to be overloaded if and only if both methods having same name but different argument types

```
class  Test{
        public void m1 (int i) {
        }
        public void m1 (long i) {
        }
}
```

2.3 Method overloading (Static Polymorphism)

In order to overload a method, the argument lists of the methods must differ in either of these

1. Number of parameters.
   add(int, int)
   add(int, int, int)
2. Data type of parameters.
   add(int, int)
   add(int, float)
3. Sequence of Data type of parameters.
   add(int, float)
   add(float, int)

INVALID CASE OF METHOD OVERLOADING
int add(int, int) float add(int, int) return type of the method must be same.

Points: Static Polymorphism is also known as compile time binding or early binding. **Static binding** happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

2.3 Method overloading (Extra Information)

Type Promotion table:

byte → short → int → long
short → int → long
int → long → float → double
float → double
long → float → double

2.3 Method overloading: Features

Method overloading
1. More than one method with same name, different prototype in same scope is called method overloading.
2. In case of method overloading, parameter must be different.
3. Method overloading is the example of compile time polymorphism.
4. Method overloading is performed within class.
5. In case of method overloading, Return type can be same or different.
6. Static methods can be overloaded which means a class can have more than one static method of same name.
7. Static binding is being used for overloaded methods

Lets’ s solve: Exp 4: Method Overloading

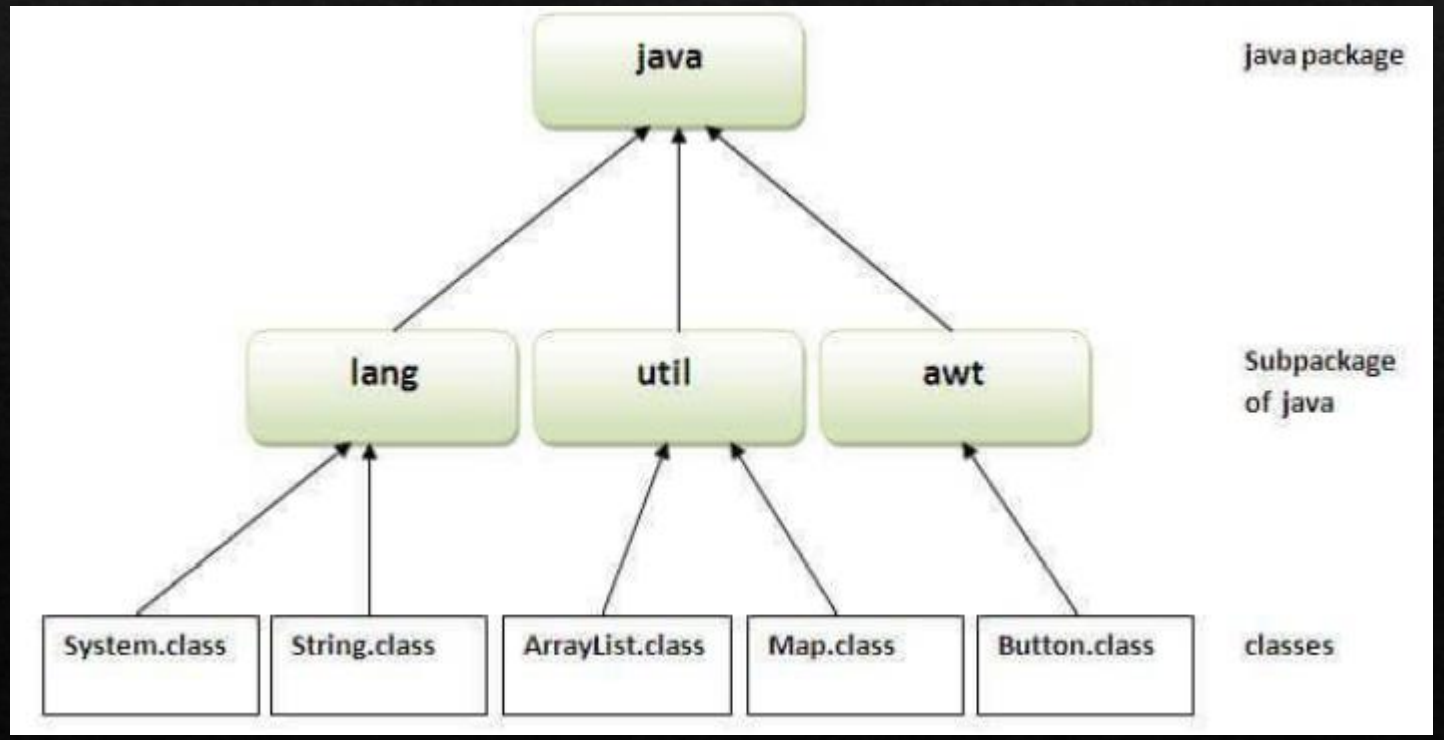
Problem Statement: Write a java program to demonstrate working of method overloading and constructor overloading in Java.

Packages

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 and 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:

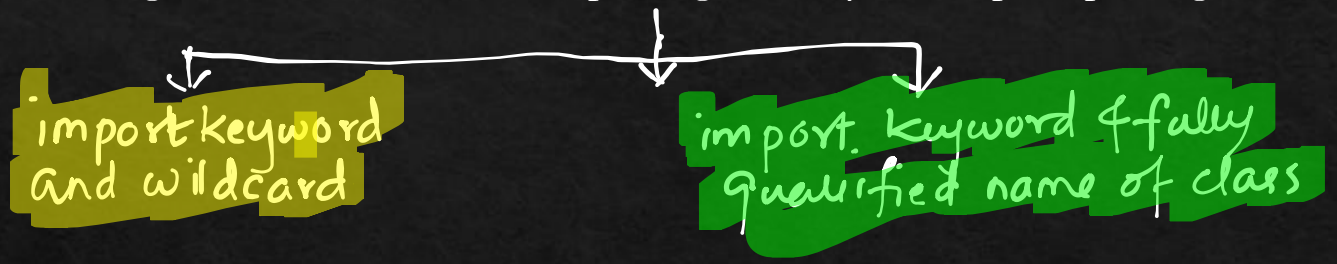
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.



Packages: User Defined : Working

The package statement should be the first line in the source file

Java import keyword is used to make classes & interface of another Package accessible to current package: 2 ways to import package are:



{When not working in IDE] : To compile the Java programs with package statements, you have to use -d option as shown below:
javac -d Destination_folder file_name.java
Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder.

```
package package1;
public class A{
  public static void main(String[] args){
        A object = new A();
        Object.display();
  }
  public void display(){
  System.out.println("Ex prgm from package1");
  }
}
```

```
package package2;
import package1.*;
public class B{
 public static void main(String[] args){
      A object = new A();
      object.display();
    }
}
```

o/p → Ex program from package 1

OR →

```
package package2;
import package1.A;
```

Lets' s solve: Exp 5: Packages

Problem Statement: Write a java program to demonstrate packages

Input/Output in Java

Program to read from a File using Buffered Reader [File as Input]

```
public class BufferedReaderFileExample {

public static void main(String[] args) throws IOException  {

    FileReader r=new FileReader("C:\\Users\\Indu
    Anoop\\Desktop\\CMPN\\myFile.txt");

      BufferedReader br=new BufferedReader(r);
      String line;
      while( (line =br.readLine()) != null){
      System.out.println(line);
      }
    }
}
```

Let's solve

Program to write output to a File

```
//To learn to write to a file

import java.io.BufferedWriter;

public class WriteOutput {

    public static void main(String[] args) throws IOException {


        FileWriter fw=new FileWriter("C:\\Users\\Indu Anoop\\Desktop\\CMPN\\myFile.txt");
        //Use below filewriter with true parameter if you want to retain previous contents of file and append new content
        //FileWriter fw=new FileWriter("C:\\Users\\Indu Anoop\\Desktop\\CMPN\\myFile.txt",true);
        BufferedWriter bw=new BufferedWriter(fw);
        //Overwriting the contents of the file
        bw.write("\n"+"hello");
        bw.close();
    }
}
```

That’s all for this module ☺