UNIT I - JAVA FUNDAMENTALS

- Java Data types
- Class – Object
- I / O Streams
- File Handling concepts
- Threads
- Applets
- Swing Framework
- Reflection

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

Topics to be covered…

- Java Class
  - Field Declaration
  - Method Declaration
- Method Overloading
- Java Objects
- Accessing Class Members
- Java Access Modifiers
- Java Package
- Object Initialization in Java
- Java constructor
- Constructor Overloading in Java

object-oriented programming

- Java is an object-oriented programming language.
- It is **based on the concept of objects**.
- These objects share two characteristics:
  - ➢state (fields/property)
  - ➢behavior (methods)
- For example: Lamp is an object
  State: on or off
  Behavior: turn on or turn off

Java Class

Java Class

- A class is a **blue print (or) prototype** that defines the variable(field) and methods, common to all objects of a certain kind
- In other words, a **class can be thought of as a user-defined data type** and **an object as a variable of that data type**
- **How to Define a class?**

```
class classname [extends superclass name]
{
   [field declarations;]
   [method declarations;]
}
```

Java Class Cont'd

- **Fields Declaration:**
  - Data is encapsulated in a class by placing data fields inside the body of the class definition.
  - These are called **instance variables**
  - They are **created whenever an object of the class is instantiated.**
  - Example:

```
Class TaxCalculator
{
      float amt=100.0f,taxRate=10.2f;
}
```

Java Class Cont'd

- **Method Declaration:**
  - A class with only data fields (& without methods that operate on that data) has no life.
  - Methods are declared inside the body of the class but immediately after the declaration of instance variables

```
ReturnType  methodName(parameter list)
{
        method-body;
}
```

Separate each parameter with ,

Example:

```
class TaxCalculator
{
        float amt=100.0f,taxRate=10.2f;
        void calcTax()
        {
                float tax;
                tax=amt*taxRate/100;
        }
}
```

•Instance variables and methods in classes are accessible by all the methods in the class, but a method cannot access the variables declared in other methods

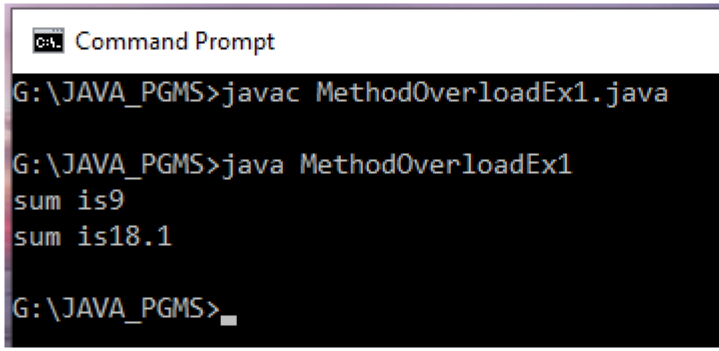
Method Overloading in Java

Method Overloading in Java

- It **is a concept** that allows to declare **multiple methods with same name but different parameters in the same class.**
- It always occur in the same class(unlike method overriding).
- Method overloading is one of the ways through which java supports polymorphism.
- Polymorphism is a concept of object oriented programming that deal with multiple forms
- Method overloading can be done by **changing number of arguments** or by **changing the data type of arguments**.
- If two or more method have same name and same parameter list **but differs in return type** can not be overloaded.
- There are two different ways of method overloading.
  - ➢ Different datatype of arguments
  - ➢ Different number of arguments

Methods with same name but different types of parameters -Example

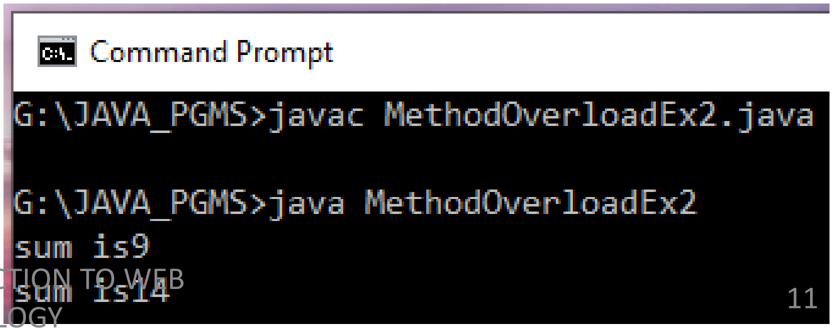
```
class MethodOverloadEx1
{
        void sum (int a, int b)
        {
                System.out.println("sum is"+(a+b)) ;
        }
        void sum (double a, double b)
        {
                System.out.println("sum is"+(a+b));
        }
        public static void main (String[] args)
        {
                MethodOverloadEx1  obj = new MethodOverloadEx1();
                obj.sum (4,5);      //sum(int a, int b) is method is called.
                obj.sum (12.3,5.8); //sum(double a, double b) is called.
        }
}
```



```
G:\JAVA_PGMS>javac MethodOverloadEx1.java

G:\JAVA_PGMS>java MethodOverloadEx1
sum is9
sum is18.1

G:\JAVA_PGMS>_
```

Method overloading by changing no. of argument - Example

```
class MethodOverloadEx2
{
        void sum (int a, int b)
        {
                System.out.println("sum is"+(a+b)) ;
        }
        void sum (int a,int b,int c)
        {
                System.out.println("sum is"+(a+b+c));
        }
        public static void main (String[] args)
        {
                MethodOverloadEx2  obj = new MethodOverloadEx2();
                obj.sum (4,5);     //sum(int a, int b) is method is called.
                obj.sum (1,6,7); //sum(int a,int b,int c) is called.
        }
}
```


Command Prompt

```
G:\JAVA_PGMS>javac MethodOverloadEx2.java

G:\JAVA_PGMS>java MethodOverloadEx2
sum is9
sum is14
```

Java Objects

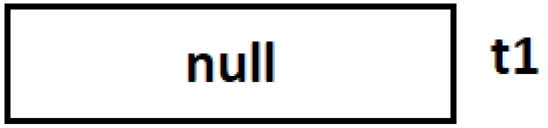
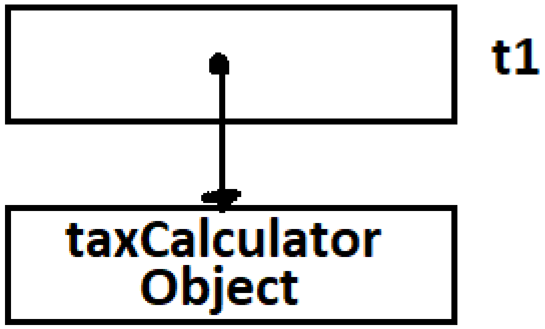
Java Objects

- Object is an **instance of a class** while class is a blueprint of an object.
- An object represents the class and consists of **properties** and **behavior**.
- Properties refer to the fields declared with in class and behavior represents to the methods available in the class.
- **Creating Objects:**
  - ➢ Object in java is essentially a block of memory that contains space to store all the instance variables
  - ➢ Creating an object is also known as **instantiating an object**
  - ➢ Objects in java are created using the **new operator**
  - ➢ new→ creates an object of the specified class and returns a reference to that object

Java Objects Cont’d

Tax Calculator t1; → declare the object

t1=new TaxCalculator() → instantiate the object

Action	**Statement**	**Result**
Declare	TaxCalculator t1	
Instantiate	t1=new TaxCalculator();	

- Both the above statements can be combined into one, **TaxCalculator t1=new TaxCalculator();**

Java Objects Cont'd

TaxCalculator t1=new TaxCalculator();

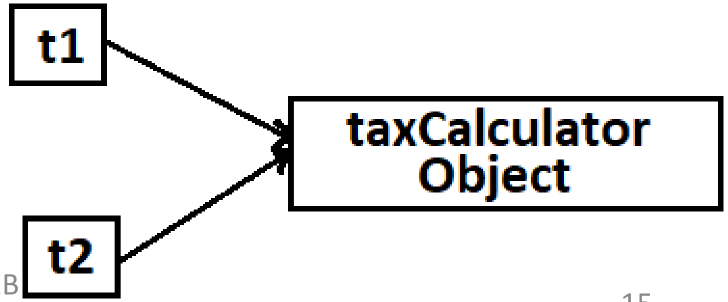
- The method TaxCalculator() is the **default Constructor** of the class
- We can create any number of objects, of class

  TaxCalculator t1=new TaxCalculator();

  TaxCalculator t2=new TaxCalculator();
- **Each object has its own copy of the instance variables** of its class
- Any change to the variable in one object have no effect on another
- It is also possible to create two or more references to the same object

TaxCalculator t1=new TaxCalculator();

TaxCalculator t2=t1;



Accessing Class Members

Accessing Class Members

- We cannot access the instance variables and the methods directly
- We must use concerned object and dot operator

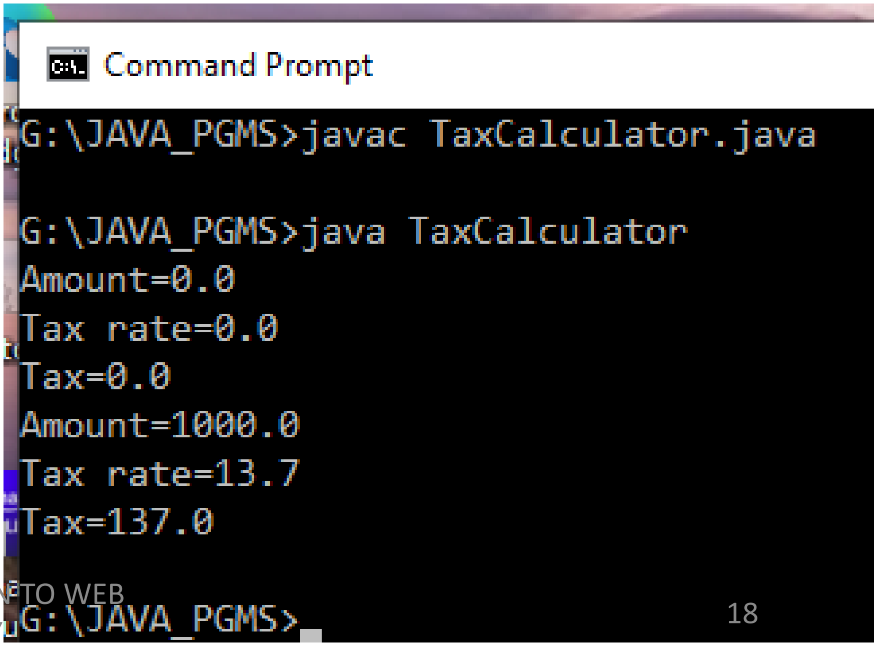
```
Objectname.variablename=value;
Objectname.methodname(parameter list);
```

- Example:

```
TaxCalculator t1=new TaxCalculator();
t1.amt=1000;
t1.calTax()
```

Object and Class Example: main within the class

```
class TaxCalculator
{
   float amt,taxRate;
   void calcTax()
    {
            float tax;
            tax=amt*taxRate/100;
            System.out.println("Amount="+amt);
            System.out.println("Tax rate="+taxRate);
            System.out.println("Tax="+tax);
    }
    public static void main(String a[])
   {
            TaxCalculator t1=new TaxCalculator();
            t1.calcTax();
            t1.amt=1000;
            t1.taxRate=13.7f;
            t1.calcTax();
   }

}
```


Command Prompt

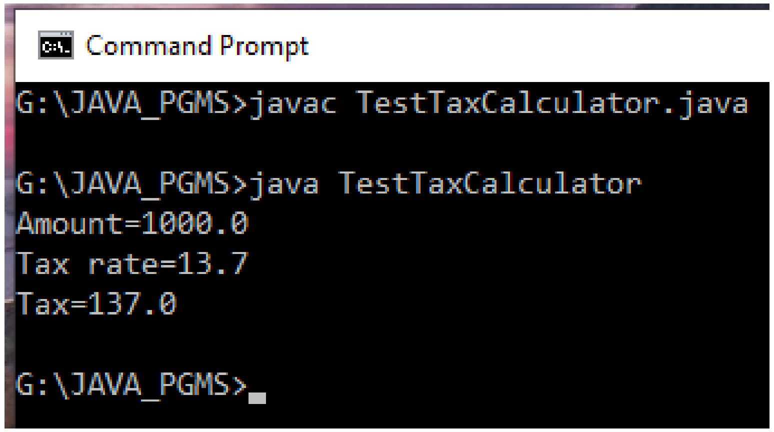
```
G:\JAVA_PGMS>javac TaxCalculator.java

G:\JAVA_PGMS>java TaxCalculator
Amount=0.0
Tax rate=0.0
Tax=0.0
Amount=1000.0
Tax rate=13.7
Tax=137.0

G:\JAVA_PGMS>
```

Object and Class: main outside the class

- In real time development, we create classes and use it from another class.
- It is a better approach than previous one.
- We can have multiple classes in different Java files or single Java file.
- If we define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

Object and Class Example: main outside the class

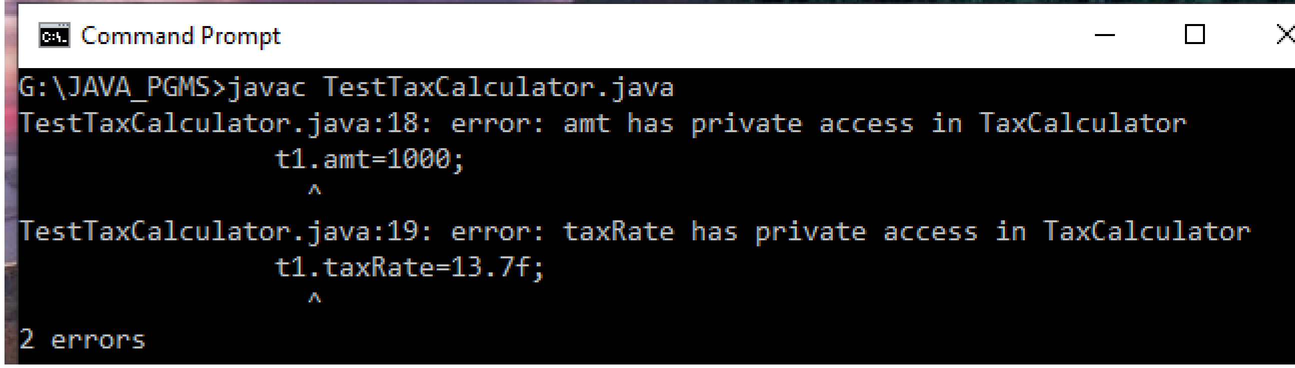
```
class TaxCalculator
{
        float amt,taxRate;
        void calcTax()
        {
                float tax;
                tax=amt*taxRate/100;
                System.out.println("Amount="+amt);
                System.out.println("Tax rate="+taxRate);
                System.out.println("Tax="+tax);
        }
}
class TestTaxCalculator
{
        public static void main(String a[])
        {
                TaxCalculator t1=new TaxCalculator();
                t1.amt=1000;
                t1.taxRate=13.7f;
                t1.calcTax();
        }
}
```



Command Prompt

```
G:\JAVA_PGMS>javac TestTaxCalculator.java

G:\JAVA_PGMS>java TestTaxCalculator
Amount=1000.0
Tax rate=13.7
Tax=137.0

G:\JAVA_PGMS>_
```

```
class TaxCalculator
{
 private float amt,taxRate;
 void calcTax()
 {
        float tax;
        tax=amt*taxRate/100;
        System.out.println("Amount="+amt);
        System.out.println("Tax rate="+taxRate);
        System.out.println("Tax="+tax);
 }
}
class TestTaxCalculator
{
  public static void main(String a[])
  {
        TaxCalculator t1=new TaxCalculator();
        t1.amt=1000;
        t1.taxRate=13.7f;
        t1.calcTax();
  }
```



```
G:\JAVA_PGMS>javac TestTaxCalculator.java
TestTaxCalculator.java:18: error: amt has private access in TaxCalculator
                t1.amt=1000;
                  ^
TestTaxCalculator.java:19: error: taxRate has private access in TaxCalculator
                t1.taxRate=13.7f;
                  ^
2 errors
```

```
}
```

Java Access Modifiers

Java Access Modifiers

- In Java, access modifiers are **used to set the accessibility (visibility)** of classes, interfaces, variables, methods, constructors, data members, and the setter methods.
- For example,

```
class Animal
 {
public void method1() {...}
private void method2() {...}
}
```

- In the above example,
  - ➢ method1 is public - This means it can be accessed by other classes.
  - ➢ method2 is private - This means it can not be accessed by other classes.
- Note the keyword public and private. These are access modifiers in Java. They are also known as **visibility modifiers**.

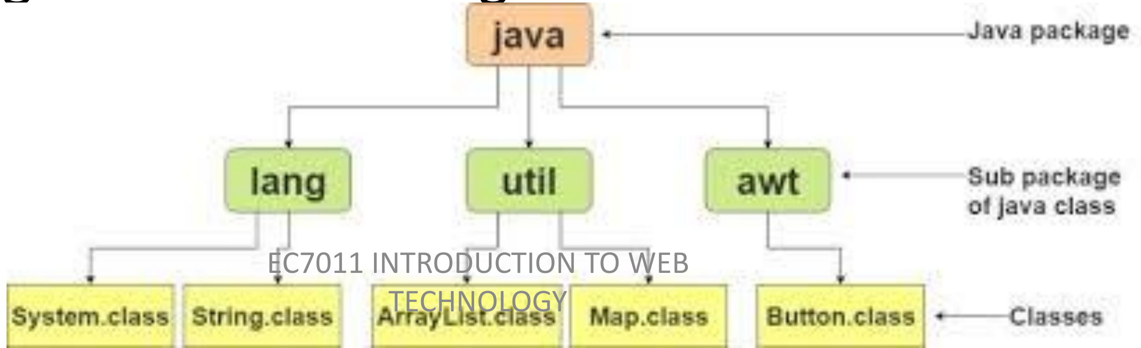
Types of Java access modifiers

Modifier	Description
private	declarations are visible within the class only
public	declarations are visible everywhere
default	declarations are visible only within the package (package private)
protected	declarations are visible within the package or all subclasses

Java Package

Java Package

- A package is simply **a container** that groups related types (Java classes, interfaces, enumerations, and annotations).
- 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.
- **<u>Advantage of Java Package</u>**
  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.



How to define a Java package?

- To define a package in Java, we use the **keyword package**.

  package packageName;

- Java uses file system directories to store packages.
- Tocreate a Java file inside another directory.
- For example:

```
└── mainpack
    └── subpack
        └── Test.java
```

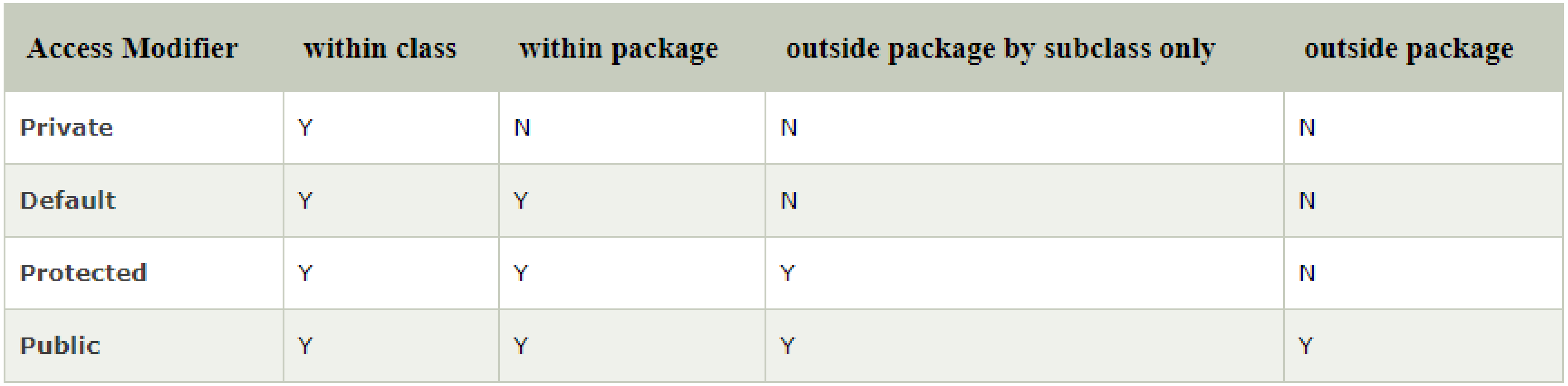
- Now, edit **Test.java** file, and at the beginning of the file, write the package statement as: package mainpack.subpack;
- **Example:**

```
package mainpack.subpack;
class Test {
        public static void main(String[] args)
       { System.out.println("Hello World!");
} }
```

Output:

```
Hello World!
```

Understanding Java Access Modifiers



Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

```
package mytest.myvisisbility;
public class MyClass
{
   public int ipub; // Visible to all
   protected int ipro; //Visible to subclass of MyClass and to other
members of mytest.myvisibility package
   int  I; //Visible only to other members of the mytest.myvisisbility
package
    private int ipri; // Visible only to MyClass Objects
}
```

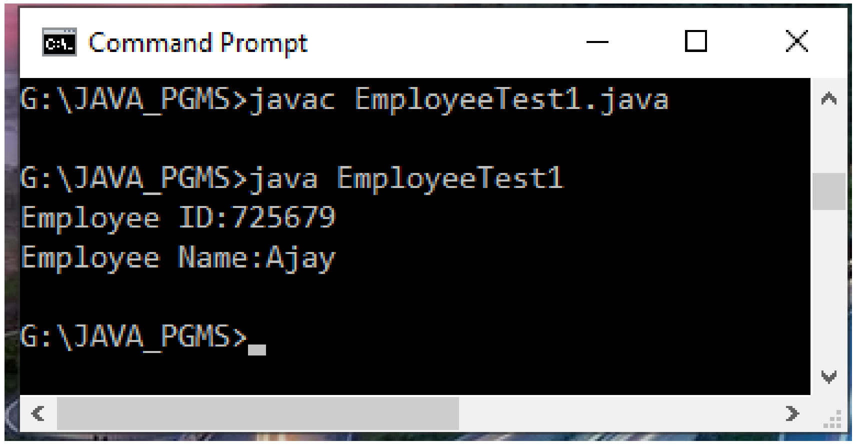
Object Initialization in Java

Object Initialization in Java

- Initializing an object **means storing data into the object**.
- There are 3 ways to initialize object in Java.
  - ➢By reference variable
  - ➢By method
  - ➢By constructor

Example -Object Initialization in Java through reference variable

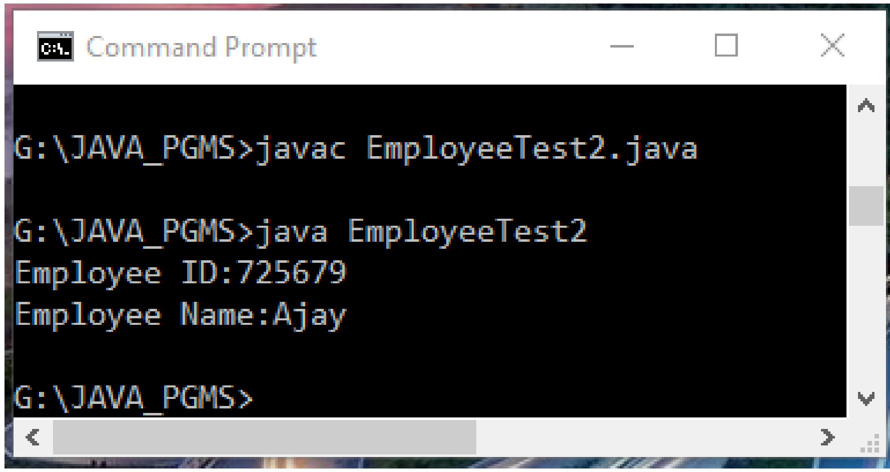
```
class Employee
{
        int empId;
        String empName;
}
class EmployeeTest1
{
        public static void main(String args[])
        {
                Employee e1=new Employee();
                e1.empId=725679;
                e1.empName="Ajay";
                System.out.println("Employee ID:"+e1.empId);
                System.out.println("Employee Name:"+e1.empName);
        }
}
```



```
G:\JAVA_PGMS>javac EmployeeTest1.java

G:\JAVA_PGMS>java EmployeeTest1
Employee ID:725679
Employee Name:Ajay

G:\JAVA_PGMS>_
```

Example -Object Initialization in Java through method

```
class Employee
{
        int empId;
        String empName;
        void assignValue(int id,String name)
        {
                empId=id;
                empName=name;
        }
}
class EmployeeTest2
{
        public static void main(String args[])
        {
                Employee e1=new Employee();
                e1.assignValue(725679,"Ajay");
                System.out.println("Employee ID:"+e1.empId);
                System.out.println("Employee Name:"+e1.empName);
        }
}
```



```
G:\JAVA_PGMS>javac EmployeeTest2.java

G:\JAVA_PGMS>java EmployeeTest2
Employee ID:725679
Employee Name:Ajay

G:\JAVA_PGMS>
```

Java constructor

Java constructor

- A constructor is a **special method** that is used **to initialize an object**.
- Every class has a constructor either implicitly or explicitly.
- If we don't declare a constructor in the class then JVM builds a default constructor for that class. This is known as **default constructor**.
- A constructor has **same name as the class name** in which it is declared.
- **Constructor must have no explicit return type**.
- Constructor in Java **can not** be **abstract, static, final or synchronized**. These modifiers are not allowed for constructor.
- **Syntax to declare constructor**

```
className (parameter-list)
{
   code-statements
 }
```

  - **className** is the name of class, as **constructor name is same as class name**.
  - **parameter-list** is optional, because constructors can be parameterize and non-parameterize as well.

Java constructor Cont'd

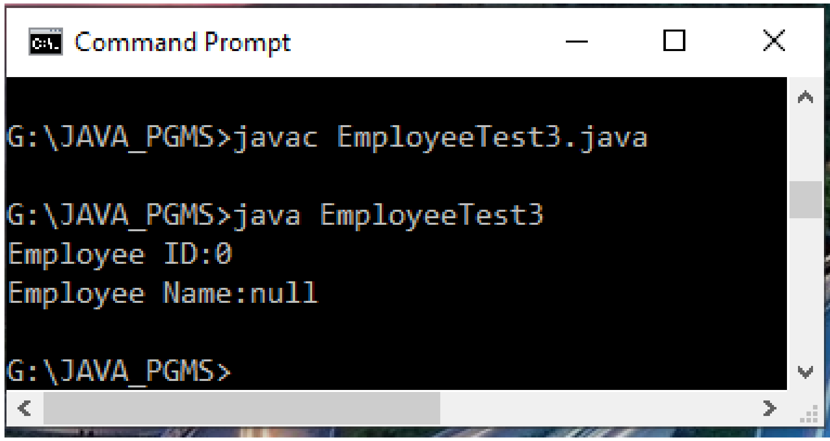
- **Types of Constructor**
- Java Supports two types of constructors:
  - ➢ Default Constructor
  - ➢ Parameterized constructor
- **Default constructor (no-arg constructor):**
  - ➢ In Java, a constructor is said to be default constructor if it **does not have any parameter**.
  - ➢ Default constructor can be **either user defined or provided by JVM**.
  - ➢ If a class does not contain any constructor then during runtime JVM generates a default constructor which is known as **system define default constructor.**
  - ➢ If a class contain a constructor with no parameter then it is known as default constructor defined by user. In this case JVM does not create default constructor.
  - ➢ The **purpose of creating constructor is to initialize states of an object**.

Java constructor Cont’d

- **Default constructor (no-arg constructor) –Cont’d:**
- When we declare a variable without assigning it an explicit value, the **Java compiler will assign a default value**
- This **default initialization applies for instance variables, not for method variables**. For variables in method, we have to initialize them explicitly.
- **Default Values for different data type:**
  - Integer numbers have default value: 0
    - for int type: 0
    - for byte type: (byte) 0
    - for short type: (short) 0
    - for long type: 0L
  - Floating point numbers have default value: 0.0
    - for float type: 0.0f
    - for double type: 0.0d
  - Boolean variables have default value: false
  - Character variables have default value: ‘\u0000’
  - Object references have default value: null

System defined default constructor – Example

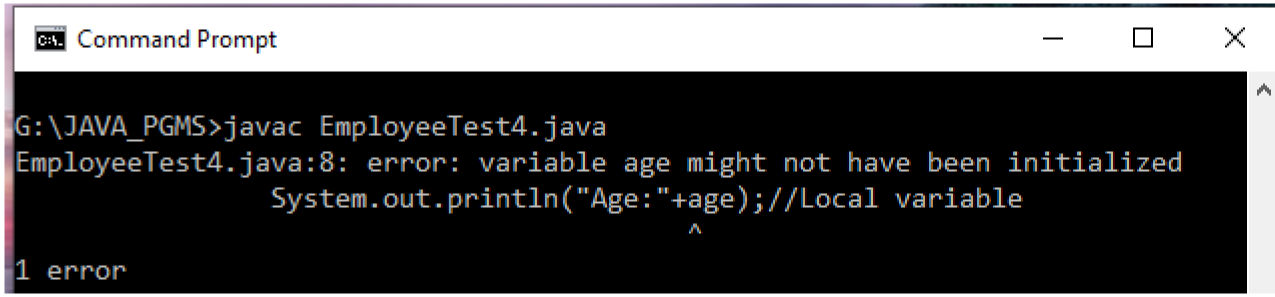
```
class Employee
{
        int empId;
        String empName;
        void display()
        {
                System.out.println("Employee ID:"+empId);
                System.out.println("Employee Name:"+empName);
        }
}
class EmployeeTest3
{
        public static void main(String args[])
        {
                Employee e1=new Employee();
                e1.display();
        }
}
```



```
G:\JAVA_PGMS>javac EmployeeTest3.java

G:\JAVA_PGMS>java EmployeeTest3
Employee ID:0
Employee Name:null

G:\JAVA_PGMS>
```

Local variables need to be initialized– Example

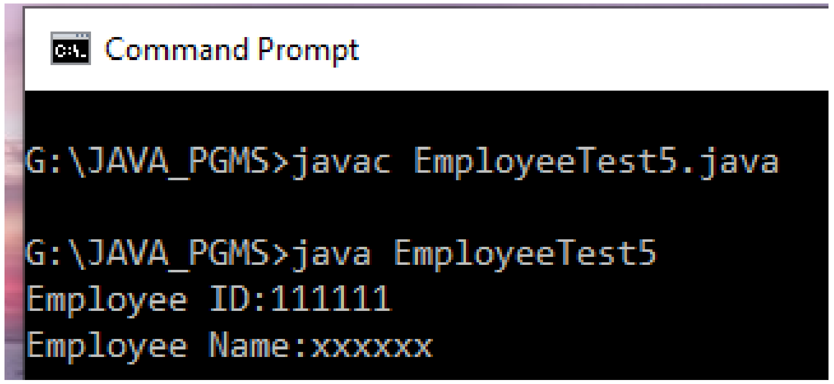
```
class Employee
{
        int empId;
        String empName;
        void display()
        {
                int age;
                System.out.println("Age:"+age);//Local variable
                System.out.println("Employee ID:"+empId);
                System.out.println("Employee Name:"+empName);
        }
}
class EmployeeTest4
{
        public static void main(String args[])
        {
                Employee e1=new Employee();
                e1.display();
        }
}
```


Command Prompt

```
G:\JAVA_PGMS>javac EmployeeTest4.java
EmployeeTest4.java:8: error: variable age might not have been initialized
                System.out.println("Age:"+age);//Local variable
                                          ^
1 error
```

User defined default constructor – Example

```
class Employee
{
	int empId;
	String empName;
	Employee()
	{
		empId=111111;
		empName="xxxxxx";
	}
	void display()
	{
		System.out.println("Employee ID:"+empId);
		System.out.println("Employee Name:"+empName);
	}
}
class EmployeeTest5
{
	public static void main(String args[])
	{
		Employee e1=new Employee();
		e1.display();
	}
}
```



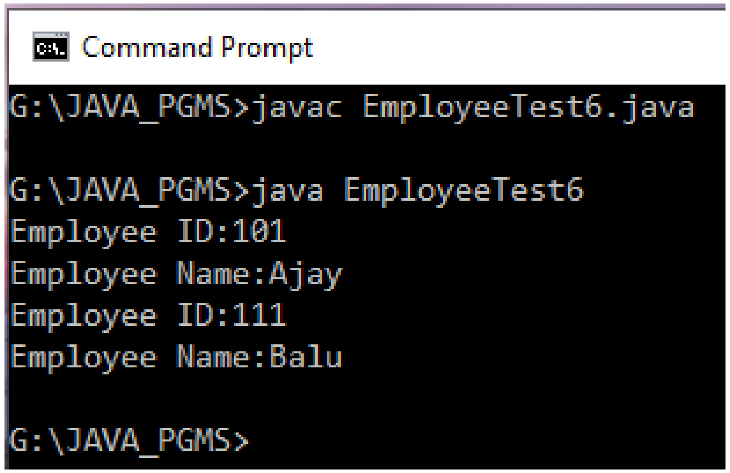
```
G:\JAVA_PGMS>javac EmployeeTest5.java

G:\JAVA_PGMS>java EmployeeTest5
Employee ID:111111
Employee Name:xxxxxx
```

Java constructor Cont’d

- Java Parameterized Constructor
- 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**

User defined parameterized constructor – Example

```
class Employee
{
        int empId;
        String empName;
        Employee(int id,String name)
        {
                empId=id;
                empName=name;
        }
        void display()
        {
                System.out.println("Employee ID:"+empId);
                System.out.println("Employee Name:"+empName);
        }
}
class EmployeeTest6
{
        public static void main(String args[])
        {
                Employee e1=new Employee(101,"Ajay");
                e1.display();
                Employee e2=new Employee(111,"Balu");
                e2.display();
        }
}
```

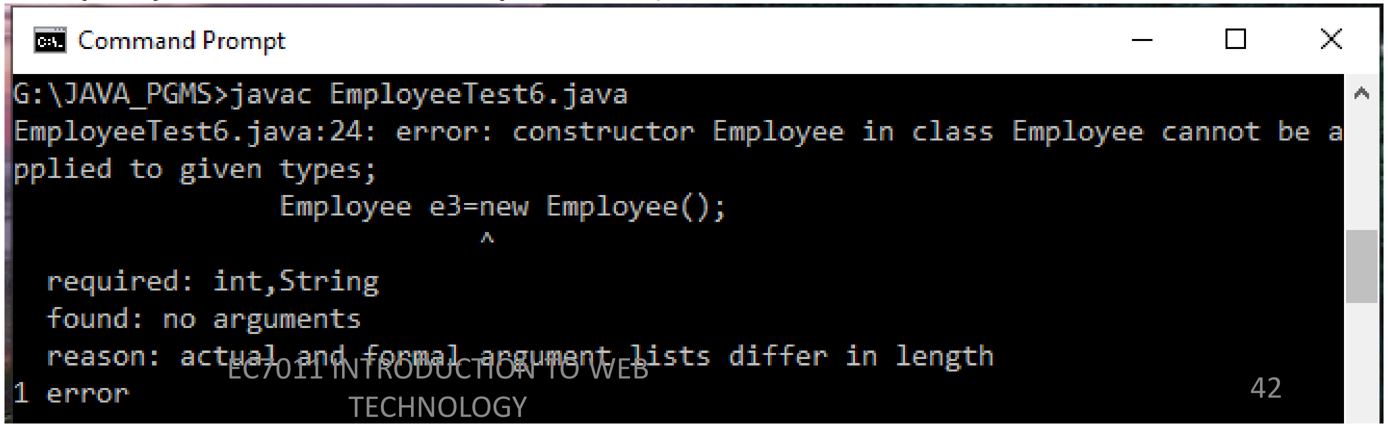


```
G:\JAVA_PGMS>javac EmployeeTest6.java

G:\JAVA_PGMS>java EmployeeTest6
Employee ID:101
Employee Name:Ajay
Employee ID:111
Employee Name:Balu

G:\JAVA_PGMS>
```

If parameterized constructor defined default constructor should be defined– Example

```
class Employee
{
 int empId;
 String empName;
 Employee(int id,String name)
 {
     empId=id;
     empName=name;
 }
 void display()
 {
   System.out.println("Employee ID:"+empId);
   System.out.println("Employee Name:"+empName);
 }
}
```

```
class EmployeeTest6
{
  public static void main(String args[])
  {
        Employee e1=new Employee(101,"Ajay");
        e1.display();
        Employee e2=new Employee(111,"Balu");
        e2.display();
        Employee e3=new Employee();
        e3.display();
   }
}
```



Command Prompt

```
G:\JAVA_PGMS>javac EmployeeTest6.java
EmployeeTest6.java:24: error: constructor Employee in class Employee cannot be a
pplied to given types;
                Employee e3=new Employee();
                            ^
  required: int,String
  found: no arguments
  reason: actual and formal argument lists differ in length
1 error
```

Constructor Overloading in Java

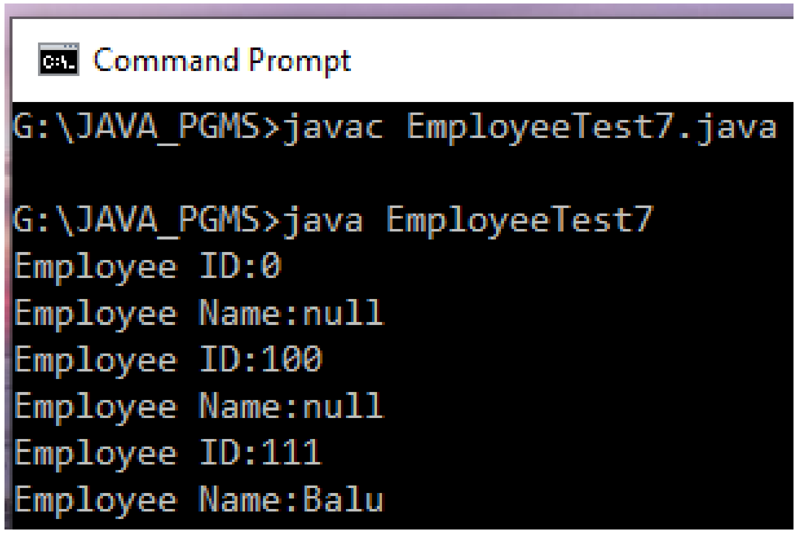
Constructor Overloading in Java

- In Java, a **constructor is just like a method** but without return type.
- It can also be overloaded like Java methods only difference is that **constructor doesn't have return type**..
- Constructor overloading in Java is a technique of **having more than one constructor with different parameter lists.**
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

Constructor Overloading Example

```
class Employee
{
 int empId;
 String empName;
 Employee()
 {
         empId=0;
         empName=null;
 }
 Employee(int id)
 {
         empId=id;
         empName=null;
 }
 Employee(int id,String name)
 {
         empId=id;
         empName=name;
 }
 void display()
 {
         System.out.println("Employee ID:"+empId);
         System.out.println("Employee Name:"+empName);
 }
}
```

```
class EmployeeTest7
{
 public static void main(String args[])
  {
         Employee e1=new Employee();
         Employee e2=new Employee(100);
         Employee e3=new Employee(111,"Balu");
         e1.display();
         e2.display();
         e3.display();
 }
}
```



Command Prompt

```
G:\JAVA_PGMS>javac EmployeeTest7.java

G:\JAVA_PGMS>java EmployeeTest7
Employee ID:0
Employee Name:null
Employee ID:100
Employee Name:null
Employee ID:111
Employee Name:Balu
```