Java | 26-5-23

atleast one class should be public in a java program.

there can’t be multiple public classes

name of public class should be same as java file

bytecode needs jre to run, jre is omnipresent.

bytecode has .class extension, its a symbolic language.

The main() method is declared static **so that JVM can call it without creating an instance of the class containing the main() method**.

public static void main(String[] args):

public: anyone can access

private: not even accesible by objects of the class.

protected: only class which is inheriting it can access it.

default: package private, which means that all members are visible within the same package but aren't accessible from other packages: package com.

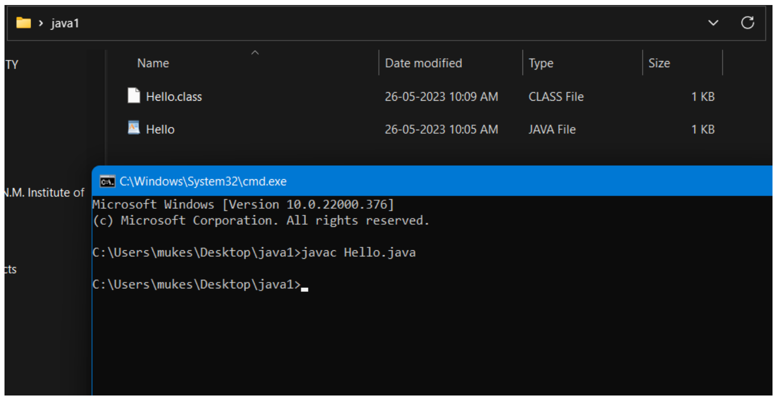
Access Specifier	in class	outside class	in package	outside package
public	y	y	y	y
private	y	n	n	n
protected	y	n	n	y
default	y	y	y	n

System.out.println("Hello world"); - println is a function written inside a System class, we need to object of System and then use it with println func. To make this simple, we use **out** object and write in one line as, System.out.println

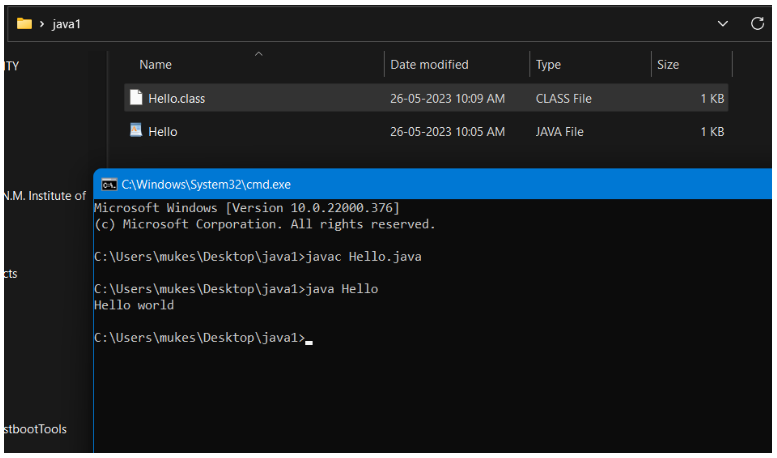
class name starts with capital letter. ex: String, Scanner are classes.

new keyword allocates memory to object.

class is non primitive data type



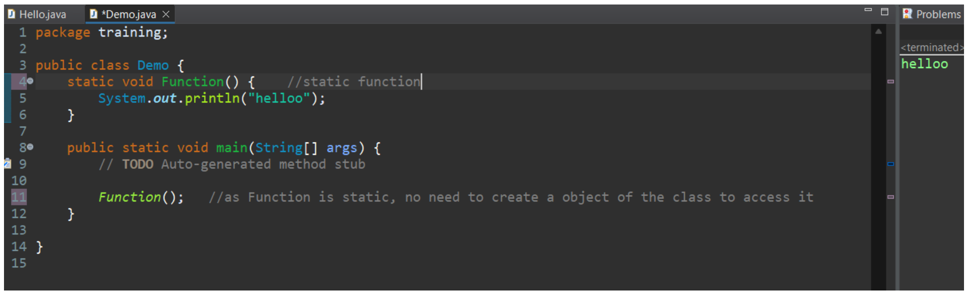
```
Microsoft Windows [Version 10.0.22000.376]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mukes\Desktop\java1>javac Hello.java

C:\Users\mukes\Desktop\java1>
```



```
Microsoft Windows [Version 10.0.22000.376]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mukes\Desktop\java1>javac Hello.java

C:\Users\mukes\Desktop\java1>java Hello
Hello world

C:\Users\mukes\Desktop\java1>
```

all .java files are present in **src**

static keyword application:

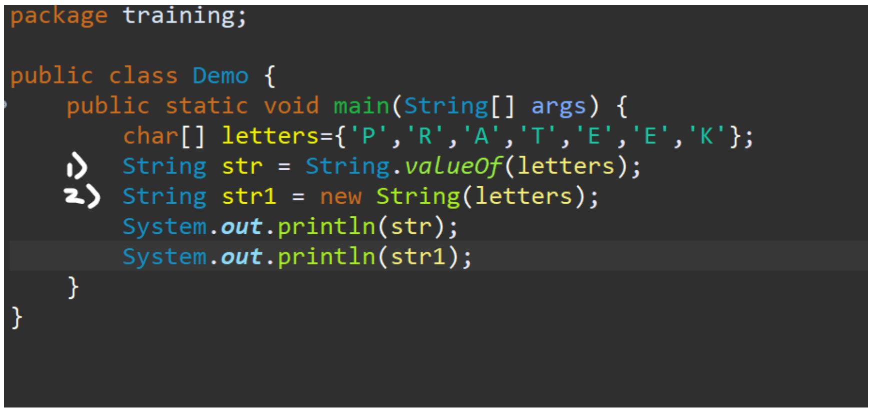


```java
package training;

public class Demo {
    static void Function() {    //static function
        System.out.println("helloo");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Function();   //as Function is static, no need to create a object of the class to access it
    }

}
```

```
helloo
```

Interview questions:

- convert character array to string in one line:



```java
package training;

public class Demo {
    public static void main(String[] args) {
        char[] letters={'P','R','A','T','E','E','K'};
1)      String str = String.valueOf(letters);
2)      String str1 = new String(letters);
        System.out.println(str);
        System.out.println(str1);
    }
}
```

- Split function to split a string

```java
String data="Prateek1sourav1is";
    String[] result=data.split("1");
    for(String i:result) {
      System.out.println(i + " ");
```

- Pattern matching

```
/**
     * illustrate pattern matching
     */
    String s1="prateek";
    String s2="PRAteek is coding";
    Pattern patt = Pattern.compile(s1, Pattern.CASE_INSENSITIVE);
      Matcher match = patt.matcher(s2);
      boolean matchFound = match.find();
      if(matchFound) {
        System.out.println("Match is found");
      } else {
        System.out.println("Match is not found");
      }
```

Object Oriented Concepts:

usage of this keyword:

```
package com.sdp.java.training;

public class Car {
  String brandName;
  int yom;
  String color;

  public Car(String brandName,int yom,String color) {
    this.brandName=brandName;
    this.yom=yom;
    this.color=color;
  }

  void printDetails() {
    System.out.println("Brand: "+brandName);
    System.out.println("YOM: "+yom);
    System.out.println("Color: "+color);
  }
}
```

```
//driver code
package com.sdp.java.training;
```

```
public class Driver {

public static void main(String[] args) {
    // TODO Auto-generated method stub
      Car c=new Car("tata",2023,"green");
      Car c1=new Car("bmw",2023,"dark blue");
      c.printDetails();
      System.out.println("------------------");
      c1.printDetails();
  }
}
```

Constructors:

```
package com.sdp.java.training;

public class Demo {
  Demo(){
    System.out.println("constructor called");
  }
  Demo(int i){
    System.out.println(i);
  }
  Demo(String s){
    System.out.println(s);
  }

  public static void main(String[] args) {
    Demo d=new Demo("prateek");
  }
}
```

Inheritance:

```
package com.sdp.java.training;

class A{
  int add(int a,int b) {
    return a+b;
  }
}

class B extends A{
  int sub(int a,int b) {
    return a-b;
  }
}

class Demo{
  public static void main(String[] args) {
    B obj=new B();
    System.out.println(obj.add(4,3));
  }
}
```

Polymorphism:

Compile	Runtime
static	dynamic
early binding	late binding
polymorphic resolves at compile time	polymorphic resolves at run time
overloading	overriding
This is also known as **method overloading**. It occurs when a class has multiple methods with the same name, but different signatures. The compiler decides which method to call based on the number and types of arguments passed to the method.	This is also known as **method overriding**. It occurs when a subclass has a method with the same name and signature as a method in its superclass. The runtime environment decides which method to call based on the actual type of the object that is being referred to.

compile time polymorphism:

```
package com.sdp.java.training;

class Addition{
  int add(int a,int b) {
    return a+b;
  }

  int add(int a,int b,int c){
    return a+b+c;
  }
}

//driver code
package com.sdp.java.training;

public class Driver {
```

```
public static void main(String[] args) {
  Addition obj=new Addition();
  System.out.println(obj.add(1,2));
  System.out.println(obj.add(1,2,3));
  }
}
```

compile time polymorphism in python

```
#simulating compile time polymorphism in python, which is not possible otherwise.
class Addition:
    def add(self, a=None, b=None, c=None):
        if(a!=None and b!=None and c!=None):
            return a+b+c
        elif(a!=None and b!=None):
            return a+b
        else:
            return a
obj=Addition()
print(obj.add(1,2))
```

runtime polymorphism:

```
package com.sdp.java.training;

class Mom{
  void cook() {
    System.out.println("indian");
  }
}

class Daughter extends Mom{
  void cook() {
    System.out.println("chinese");
  }
}

//driver code
package com.sdp.java.training;

public class Driver {

public static void main(String[] args) {
  Mom m=new Mom();
  Daughter d=new Daughter();
//  m.cook();
  d.cook();
  }
}
```

Abstraction:

Abstract methods do not specify a body

we cant create object of a abstracted class

partially abstract class: when class is abstract but not all functions are abstract.

```
package com.sdp.java.training;

abstract class Animal{ //hidden from main function, i.e abstraction
  abstract void eat();
  void run() {
    System.out.println("running....");
  }
}
class Tiger extends Animal{
  void eat() {
    System.out.println("Tiger is eating");
  }
}

//driver code
package com.sdp.java.training;

public class Driver {

public static void main(String[] args) {
  Tiger t=new Tiger();
  t.eat();
  t.run();
  }
}
```

100% abstraction in java:

interface: a class whose contents are 100% hidden

```
package com.sdp.java.training;

interface Animal{
  void eat();
  void run();
}
class Elephant implements Animal{
  public void eat() {  //access specifier public is required
    System.out.println("eating...");
  }
  public void run() {  //access specifier public is required
    System.out.println("running...");
  }
}
//driver code
package com.sdp.java.training;

public class Driver {

public static void main(String[] args) {
  Elephant e=new Elephant();
  e.eat();
  e.run();
  }
}
```

```
package com.sdp.java.training;
//multiple implement:
interface Animal{
  void eat();
  void run();
}
interface Mammal{
  void eat();
  void run();
}
class Elephant implements Animal{
  public void eat() {  //access specifier public is required
    System.out.println("elephant eating...");
  }
  public void run() {  //access specifier public is required
    System.out.println("elephant running...");
  }
}
class Penguin implements Animal,Mammal{
  public void eat() {  //access specifier public is required
    System.out.println("penguin eating...");
  }
  public void run() {  //access specifier public is required
    System.out.println("penguin running...");
  }
}
//driver code
package com.sdp.java.training;

public class Driver {

public static void main(String[] args) {
  Elephant e=new Elephant();
  e.eat();
  e.run();
  Penguin p=new Penguin();
  p.eat();
  p.run();
  }
}
```

Encapsulation:

using getters and setters

```
package com.sdp.java.training;

class student{
  private String name;
  private int usn;

  void setName(String name) {
    this.name=name;
  }

  String getName() {
    return name;
  }

}

public class Demo {

public static void main(String[] args) {
  student s=new student();
  s.setName("prateek sourav");
  String n=s.getName();
  System.out.println(n);
  }
}
```