Java Inner Classes

History:

- At 1.0 v of java Developers found GUI Bugs.
- To remove this error they introduced INNER CLASSES on 1.1v.
- After this inner classes introduced, it have advantages for developers to Write code in simple way.
- Inside a class we can declare another class i.e., Inner Class

Where we use ?

- With out existing one type of object there is no chance of another type of object, then we go for Inner Classes

Example:

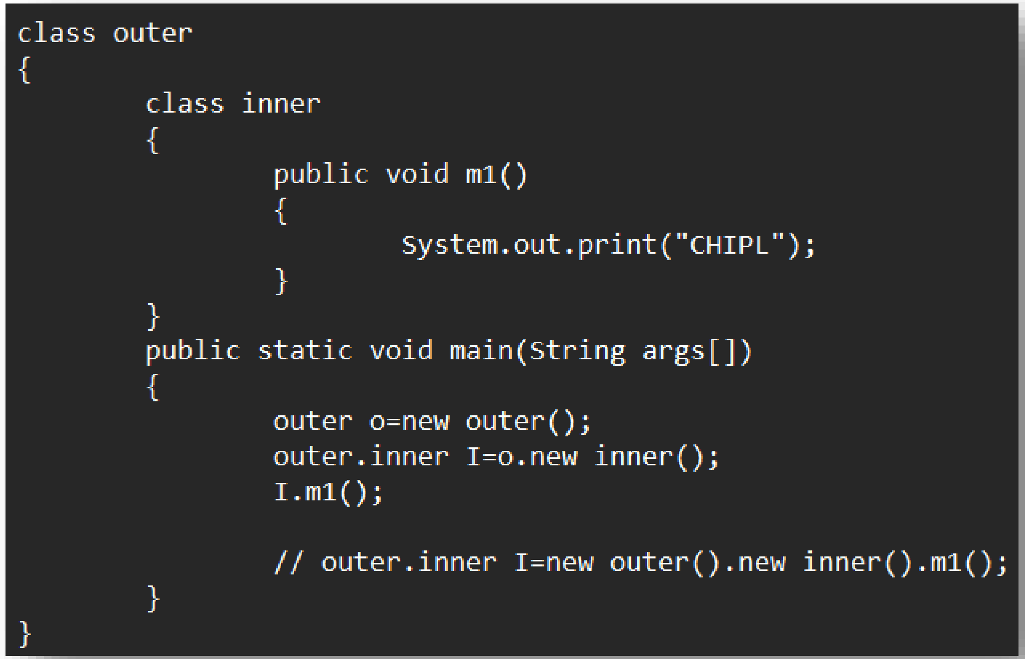
Without existing car object there is no Engine object.

Based on position declaration & behaviour:

- Normal / Regular Inner Classes
- Method Local Inner Classes
- Anonymous Inner Classes
- Static Nested Classes

Normal / Regular Inner Classes

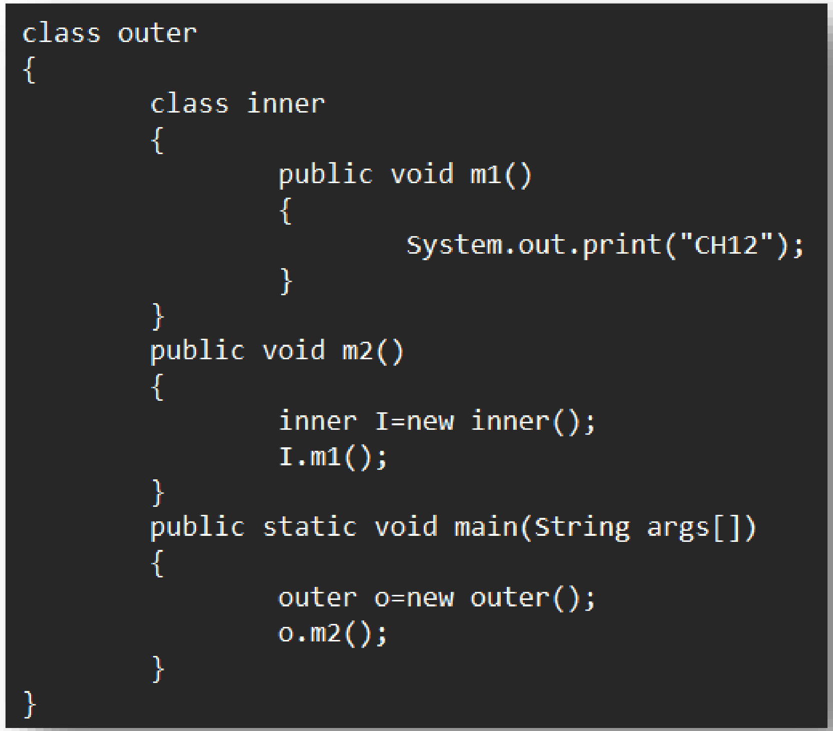
- Declaring any named class directly inside class without static modifier such type of classes is called Normal / Regular Inner Classes



```java
class outer
{
        class inner
        {
                public void m1()
                {
                        System.out.print("CHIPL");
                }
        }
        public static void main(String args[])
        {
                outer o=new outer();
                outer.inner I=o.new inner();
                I.m1();

                // outer.inner I=new outer().new inner().m1();
        }
}
```

Output:

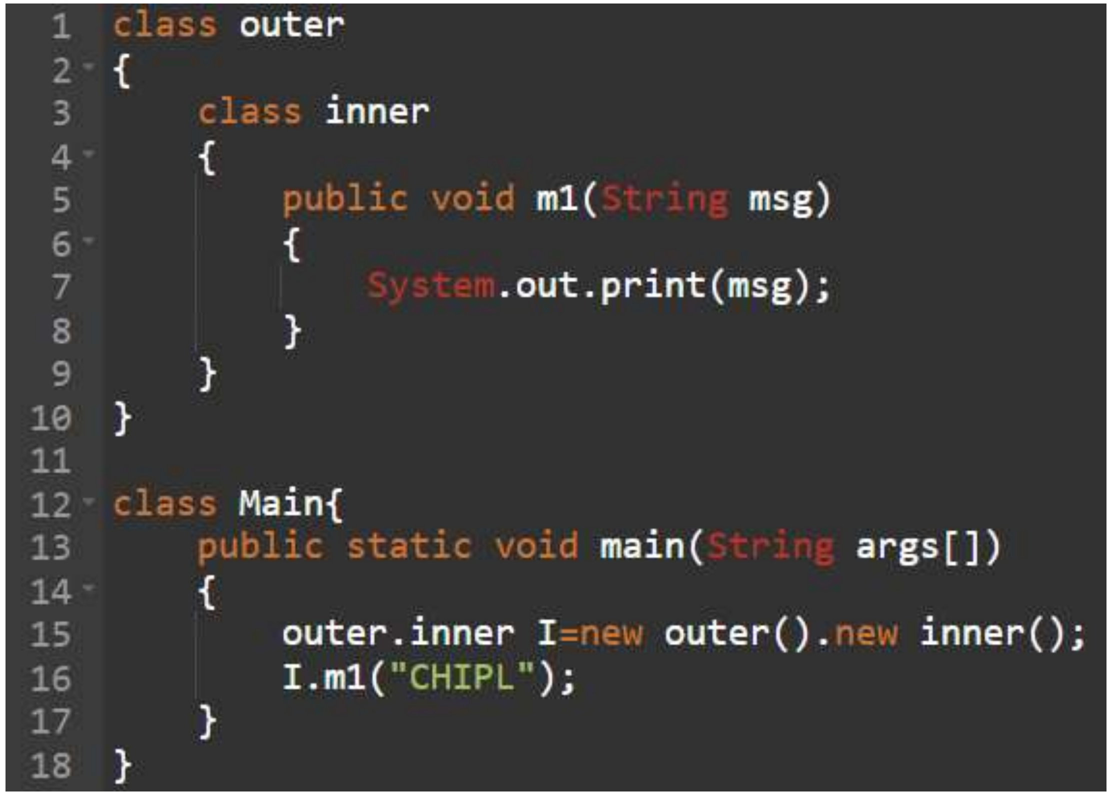
```
CHIPL
```



```
class outer
{
        class inner
        {
                public void m1()
                {
                        System.out.print("CH12");
                }
        }
        public void m2()
        {
                inner I=new inner();
                I.m1();
        }
        public static void main(String args[])
        {
                outer o=new outer();
                o.m2();
        }
}
```

Output:

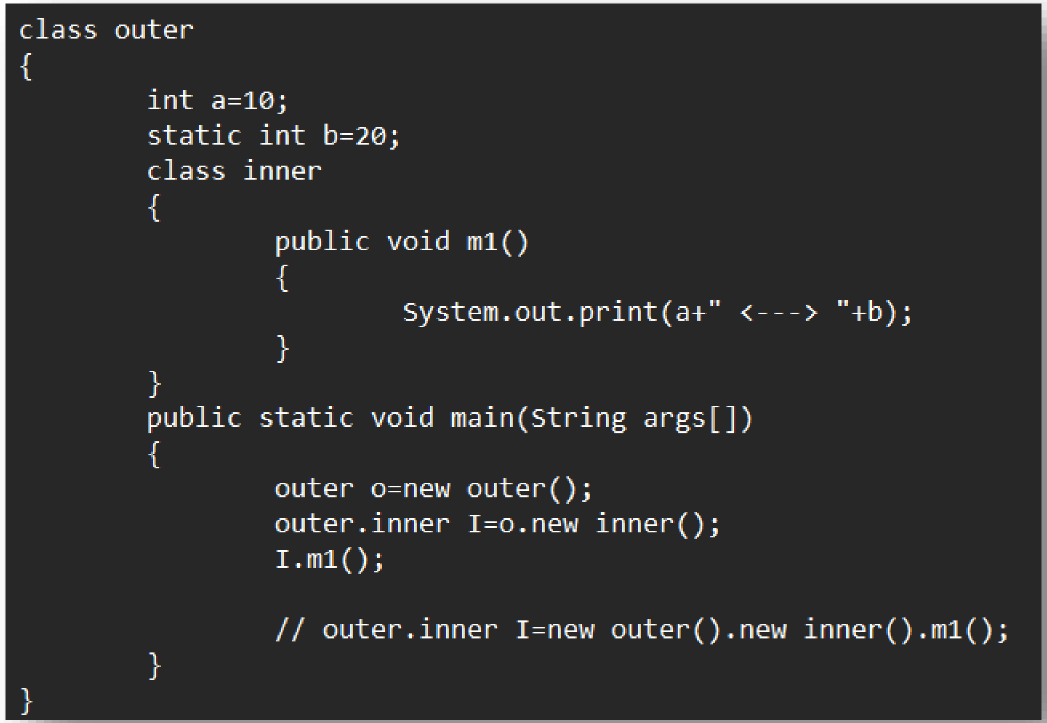
CH12



```
class outer
{
    class inner
    {
        public void m1(String msg)
        {
            System.out.print(msg);
        }
    }
}

class Main{
    public static void main(String args[])
    {
        outer.inner I=new outer().new inner();
        I.m1("CHIPL");
    }
}
```

Output:

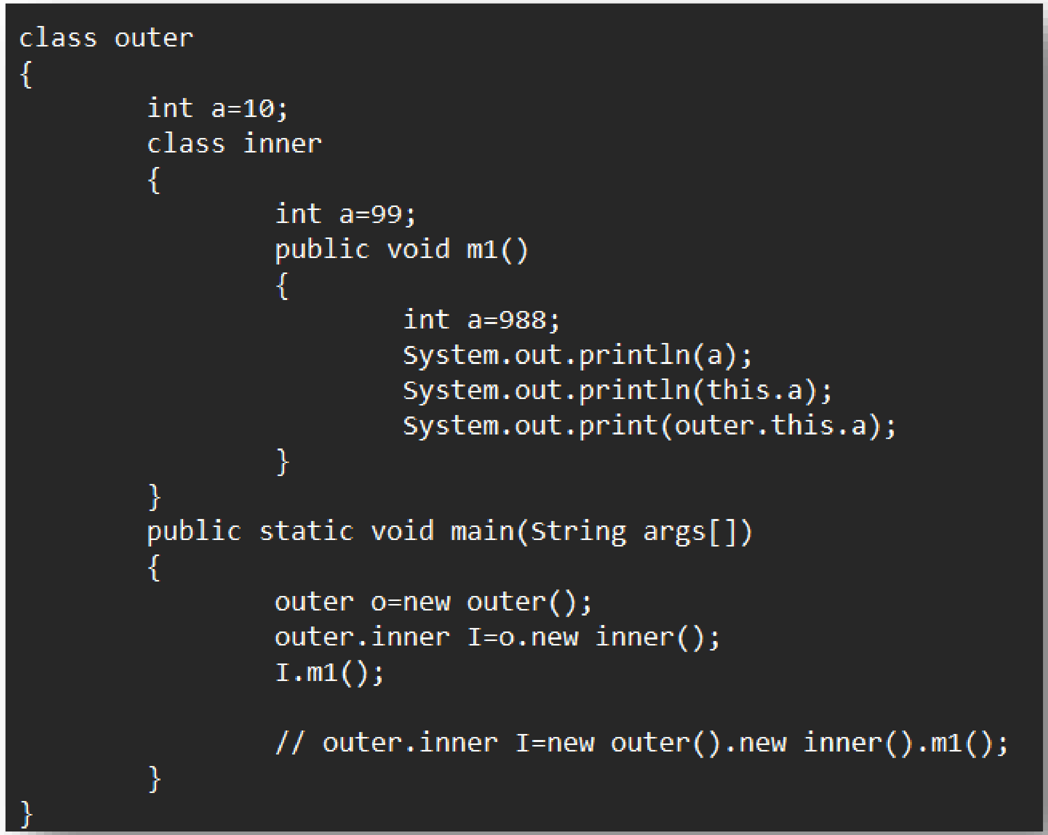
CHIPL



```
class outer
{
        int a=10;
        static int b=20;
        class inner
        {
                public void m1()
                {
                        System.out.print(a+" <---> "+b);
                }
        }
        public static void main(String args[])
        {
                outer o=new outer();
                outer.inner I=o.new inner();
                I.m1();

                // outer.inner I=new outer().new inner().m1();
        }
}
```

Output:

```
10 < --- > 20
```



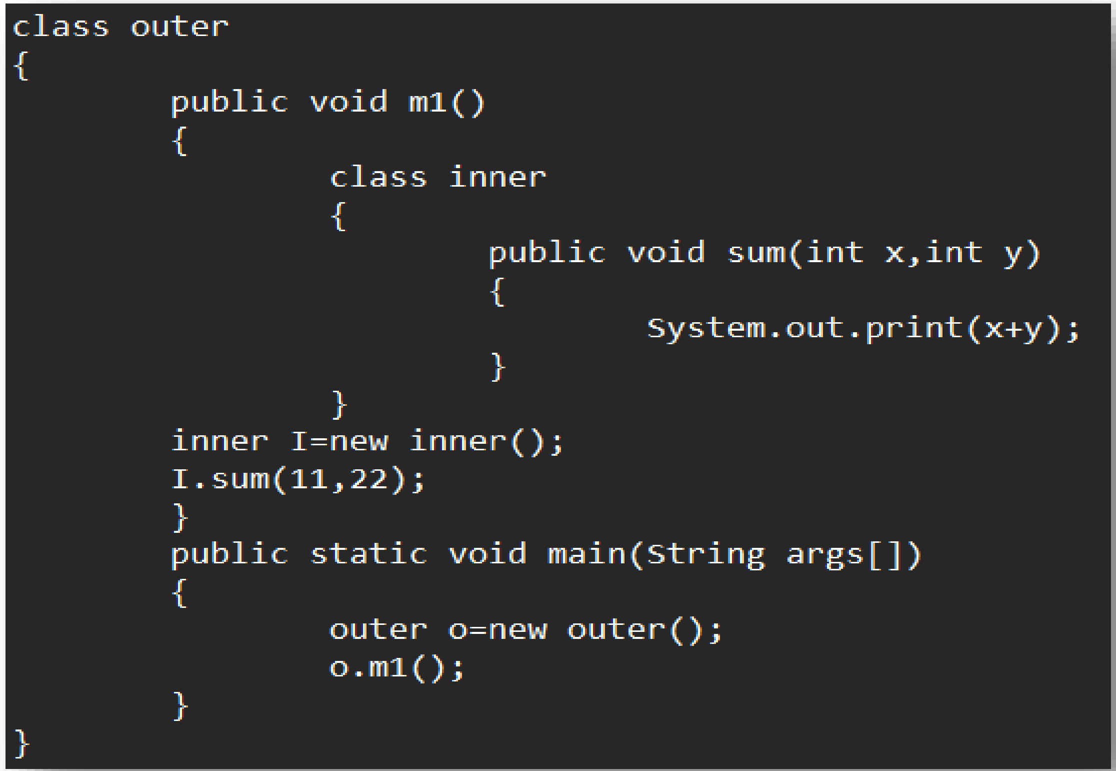
```
class outer
{
        int a=10;
        class inner
        {
                int a=99;
                public void m1()
                {
                        int a=988;
                        System.out.println(a);
                        System.out.println(this.a);
                        System.out.print(outer.this.a);
                }
        }
        public static void main(String args[])
        {
                outer o=new outer();
                outer.inner I=o.new inner();
                I.m1();

                // outer.inner I=new outer().new inner().m1();
        }
}
```

Output:

```
988
99
10
```

Method Local Inner Classes

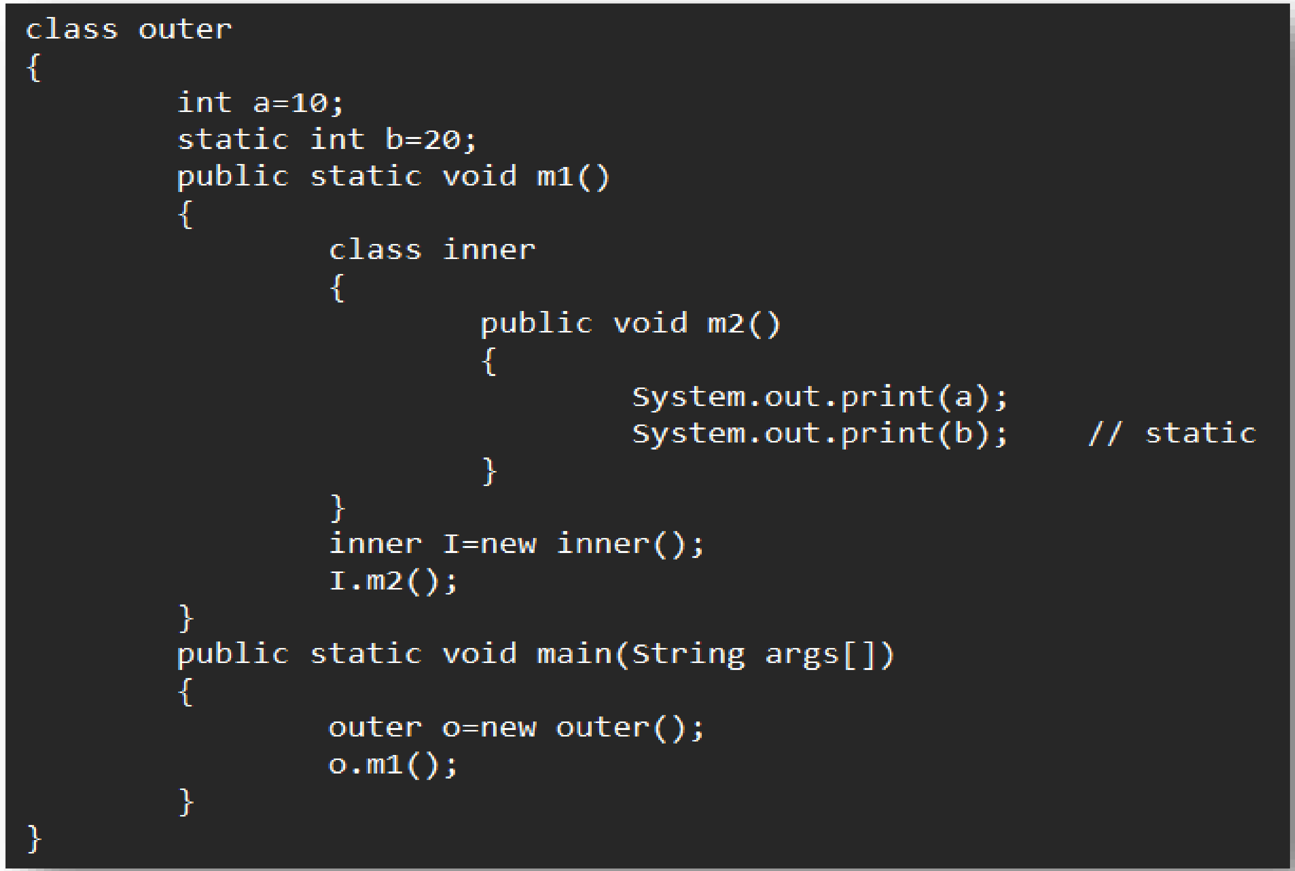
- Main purpose is to define method specific repeatedly required functionality



```
class outer
{
        public void m1()
        {
                class inner
                {
                        public void sum(int x,int y)
                        {
                                System.out.print(x+y);
                        }
                }
        inner I=new inner();
        I.sum(11,22);
        }
        public static void main(String args[])
        {
                outer o=new outer();
                o.m1();
        }
}
```

Output:

```
33
```



```
class outer
{
        int a=10;
        static int b=20;
        public static void m1()
        {
                class inner
                {
                        public void m2()
                        {
                                System.out.print(a);
                                System.out.print(b);    // static
                        }
                }
                inner I=new inner();
                I.m2();
        }
        public static void main(String args[])
        {
                outer o=new outer();
                o.m1();
        }
}
```

Output:

Error at a because a is non static used in static method

Before Going to Anonymous Inner Class

What is Functional Interface:

- A **functional interface** is an interface that contains only one abstract method. They can have only one functionality to exhibit

Example: Runnable Interface → Thread Concept

What is Marker Interface:

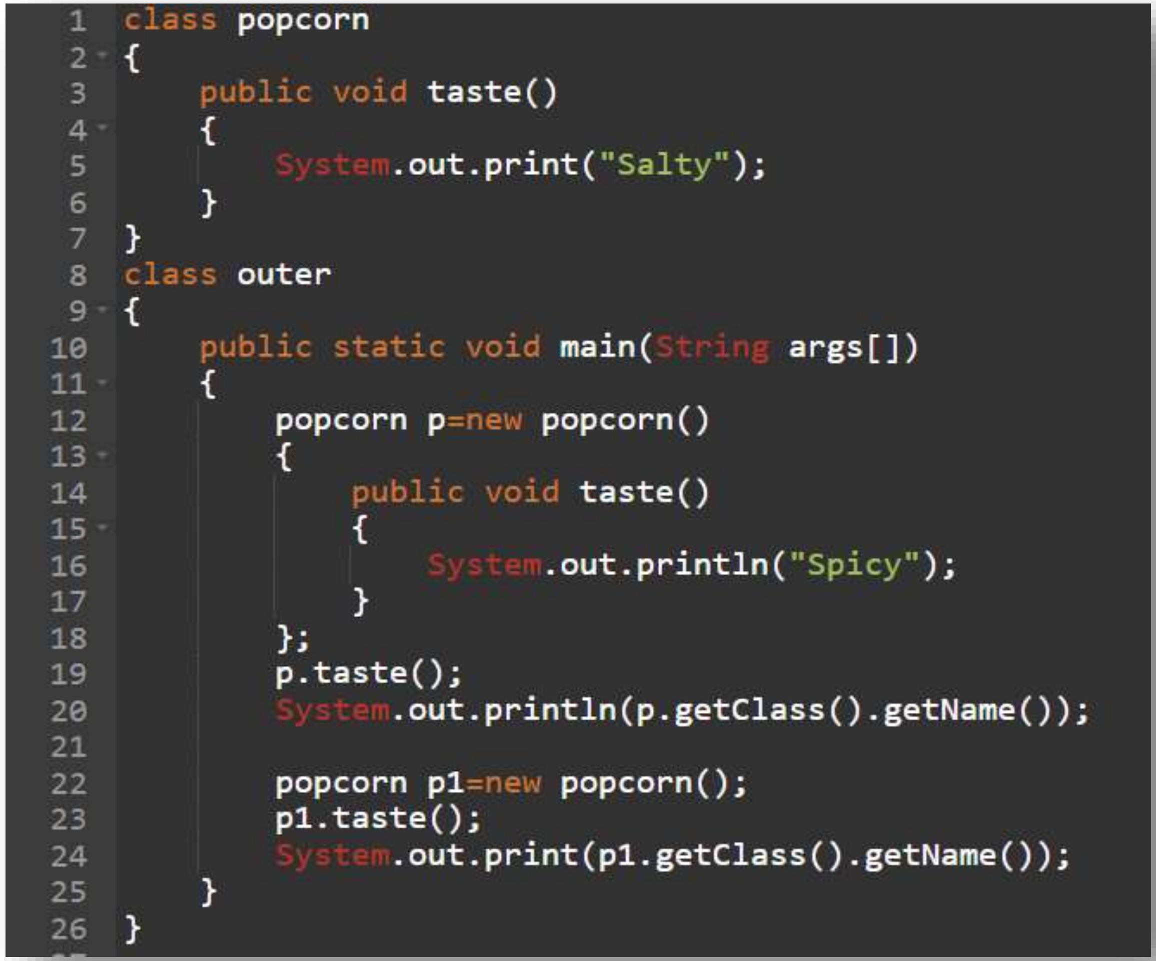
- A **Marker Interface** is an empty interface (no field or methods)

Example: Serializable, Cloneable → All these interfaces are empty interfaces

Anonymous Inner Class:

- Used for Instant Use (One time usage)
- Anonymous class does not contain any Class Name
- Anonymous class extends a class
- Anonymous class extends a Interface
- Advanced Concept is “Lambda expression”

Adding Class for Anonymous class

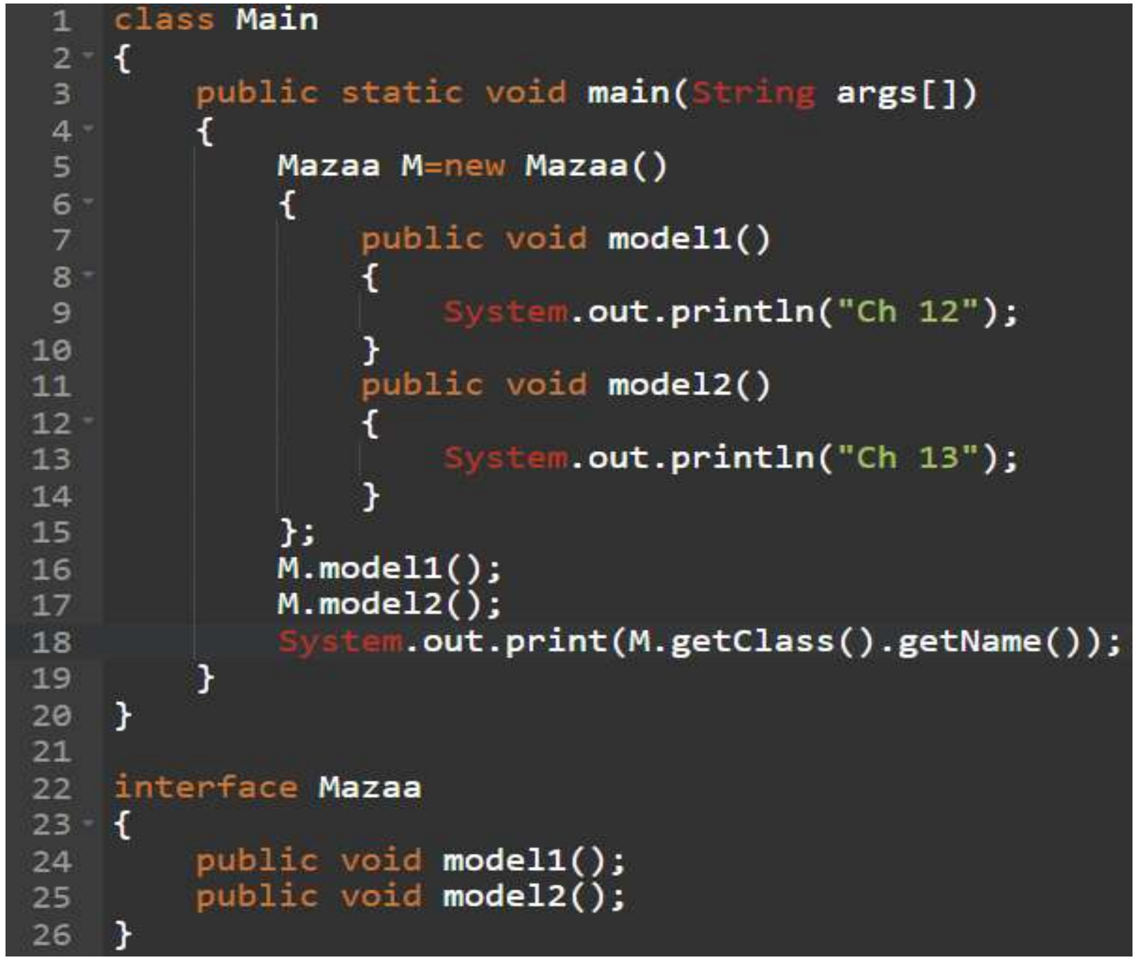


```
class popcorn
{
    public void taste()
    {
        System.out.print("Salty");
    }
}
class outer
{
    public static void main(String args[])
    {
        popcorn p=new popcorn()
        {
            public void taste()
            {
                System.out.println("Spicy");
            }
        };
        p.taste();
        System.out.println(p.getClass().getName());

        popcorn p1=new popcorn();
        p1.taste();
        System.out.print(p1.getClass().getName());
    }
}
```

Output:

```
Spicy
outer$1
Saltypopcorn
```

Adding interface for Anonymous class



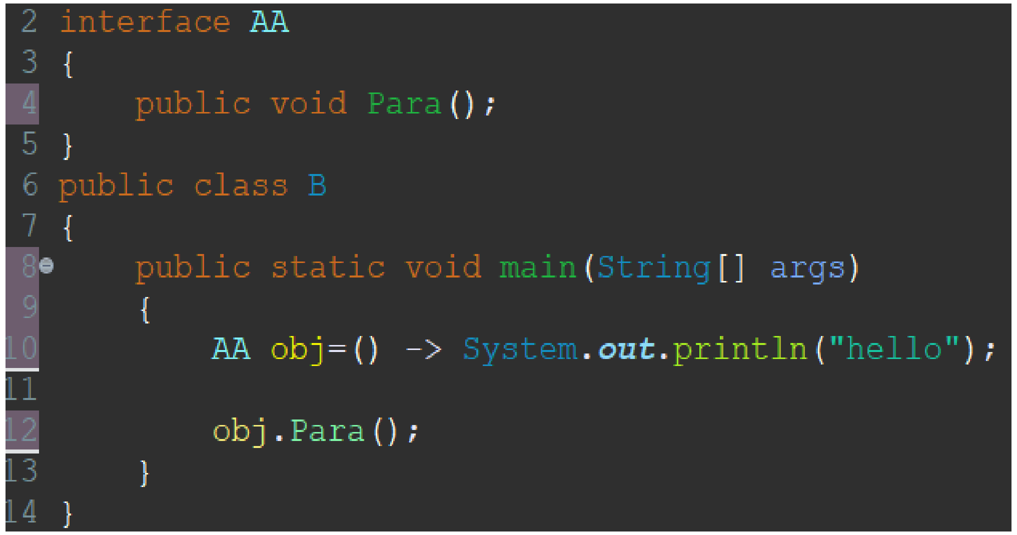
```java
class Main
{
    public static void main(String args[])
    {
        Mazaa M=new Mazaa()
        {
            public void model1()
            {
                System.out.println("Ch 12");
            }
            public void model2()
            {
                System.out.println("Ch 13");
            }
        };
        M.model1();
        M.model2();
        System.out.print(M.getClass().getName());
    }
}

interface Mazaa
{
    public void model1();
    public void model2();
}
```

Output:

```
Ch 12
Ch 13
Main$1
```

Lambda Expression:

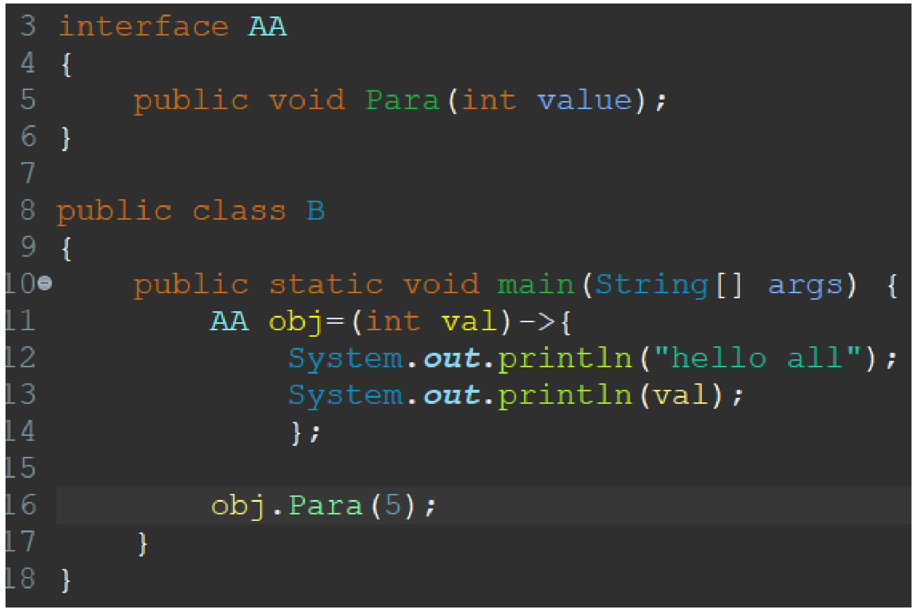
- Lambda Expressions were added in Java 8.
- Used When we have One method only.
- Lambda Expression works with only Functional Interface



```java
interface AA
{
    public void Para();
}
public class B
{
    public static void main(String[] args)
    {
        AA obj=() -> System.out.println("hello");

        obj.Para();
    }
}
```

Output:

```
hello
```

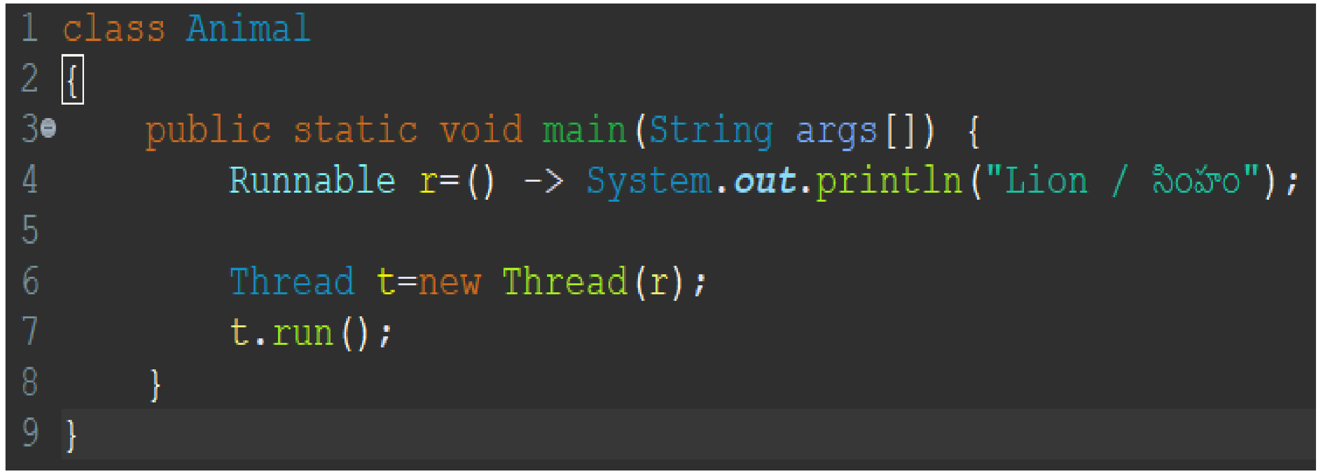


```java
interface AA
{
    public void Para(int value);
}

public class B
{
    public static void main(String[] args) {
        AA obj=(int val)->{
            System.out.println("hello all");
            System.out.println(val);
            };

        obj.Para(5);
    }
}
```

Output:

```
hello all
5
```

Using Predefined Runnable Interface:

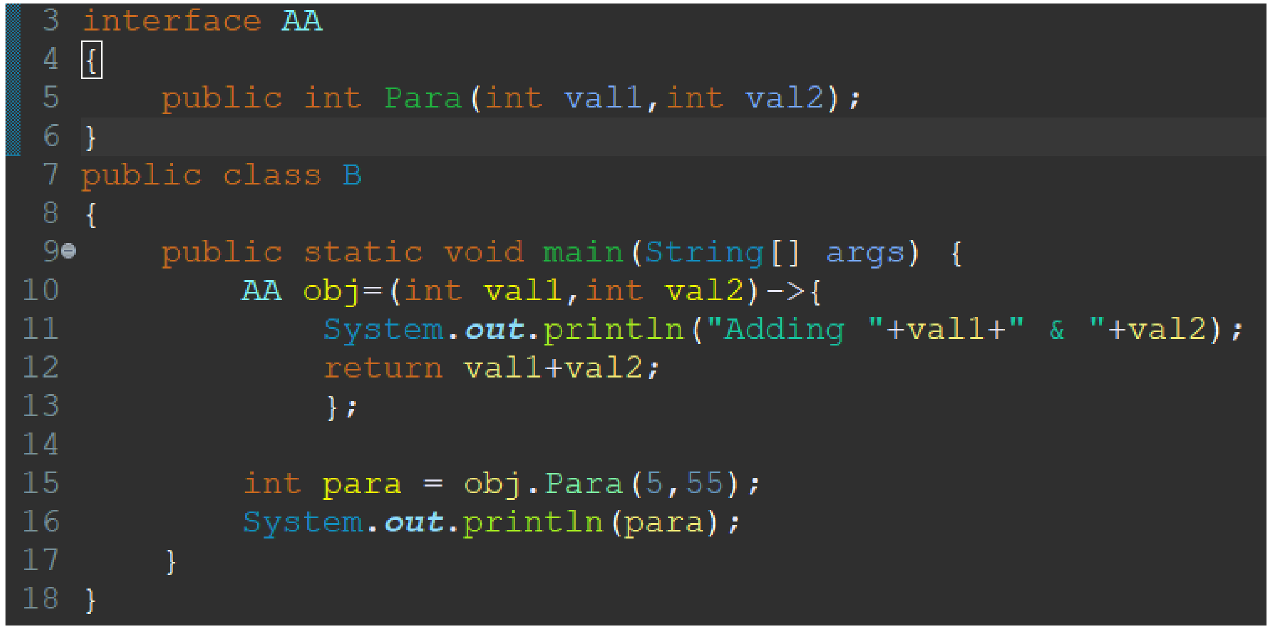


```
class Animal
{
    public static void main(String args[]) {
        Runnable r=() -> System.out.println("Lion / సింహం");

        Thread t=new Thread(r);
        t.run();
    }
}
```

Output:

```
Lion / సింహం
```

Adding return type:



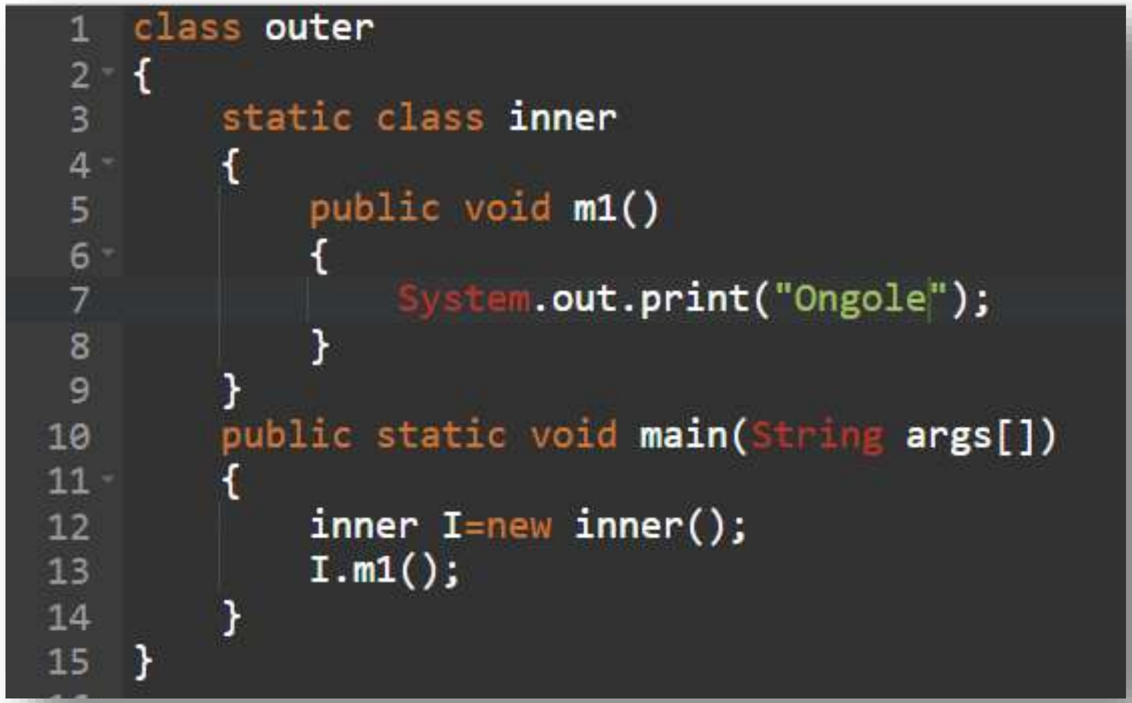
```
interface AA
{
    public int Para(int val1,int val2);
}
public class B
{
    public static void main(String[] args) {
        AA obj=(int val1,int val2)->{
            System.out.println("Adding "+val1+" & "+val2);
            return val1+val2;
            };

        int para = obj.Para(5,55);
        System.out.println(para);
    }
}
```

Output:

```
Adding 5 & 55
60
```

Static Nested Classes

- Define nested classes as static modifier is known as Static Nested Classes
- Not Strongly associated with outer class object

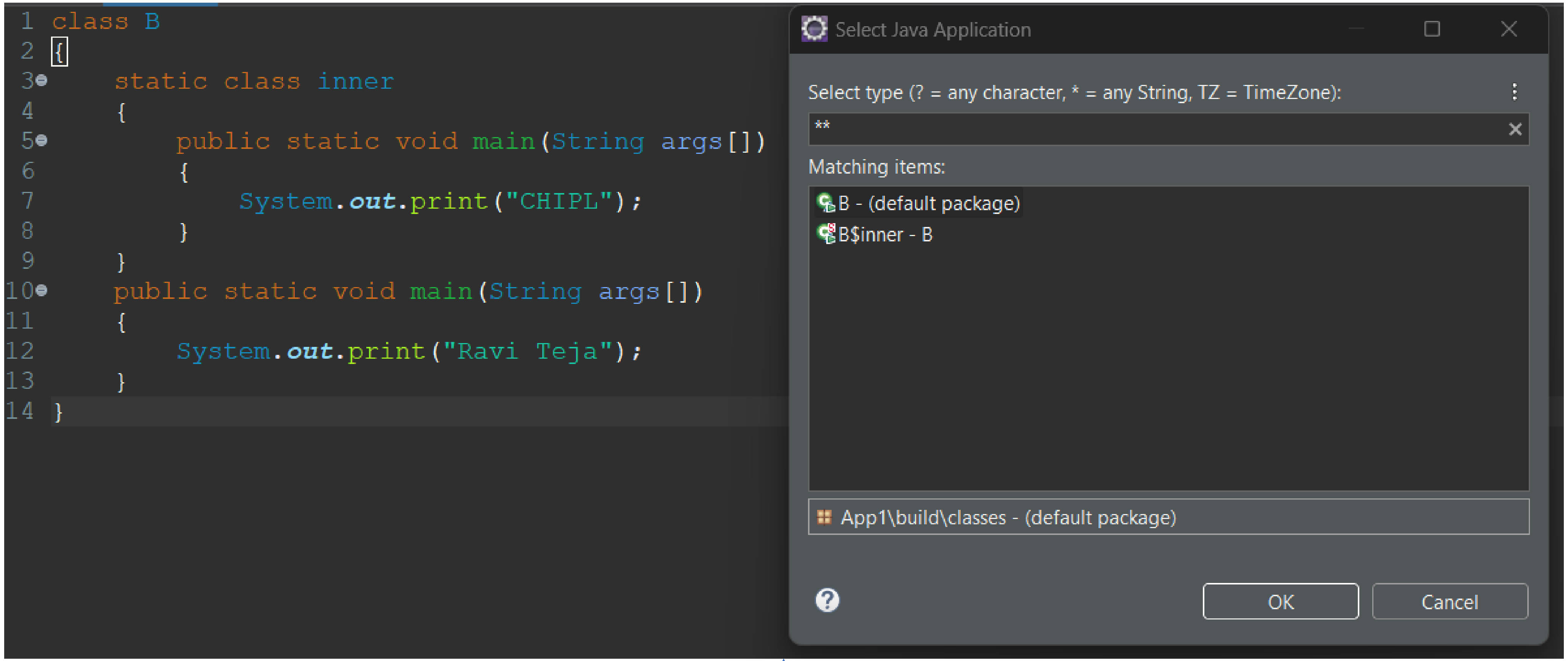


```java
class outer
{
	static class inner
	{
		public void m1()
		{
			System.out.print("Ongole");
		}
	}
	public static void main(String args[])
	{
		inner I=new inner();
		I.m1();
	}
}
```

Output:

```
Ongole
```

- You can also write 2 main methods in this static nested class



```java
class B
{
    static class inner
    {
        public static void main(String args[])
        {
            System.out.print("CHIPL");
        }
    }
    public static void main(String args[])
    {
        System.out.print("Ravi Teja");
    }
}
```

Select Java Application

Select type (? = any character, * = any String, TZ = TimeZone):

**

Matching items:

B - (default package)
B$inner - B

App1\build\classes - (default package)

Output Cmd:

```
javac outer.java
java outer          --- >    Ravi Teja
java outer$inner    --- >    CHIPL
```

Output Console:

```
B-(default package)    --- >    Ravi Teja
B$inner – B            --- >    CHIPL
```

Thank
You