Object Oriented Programming

Java™

Unit 9

Lambda Expressions

Basic of lambda expression: What is lambda expression?, Need for Lambda Expression, Type Inference

What is lambda expression

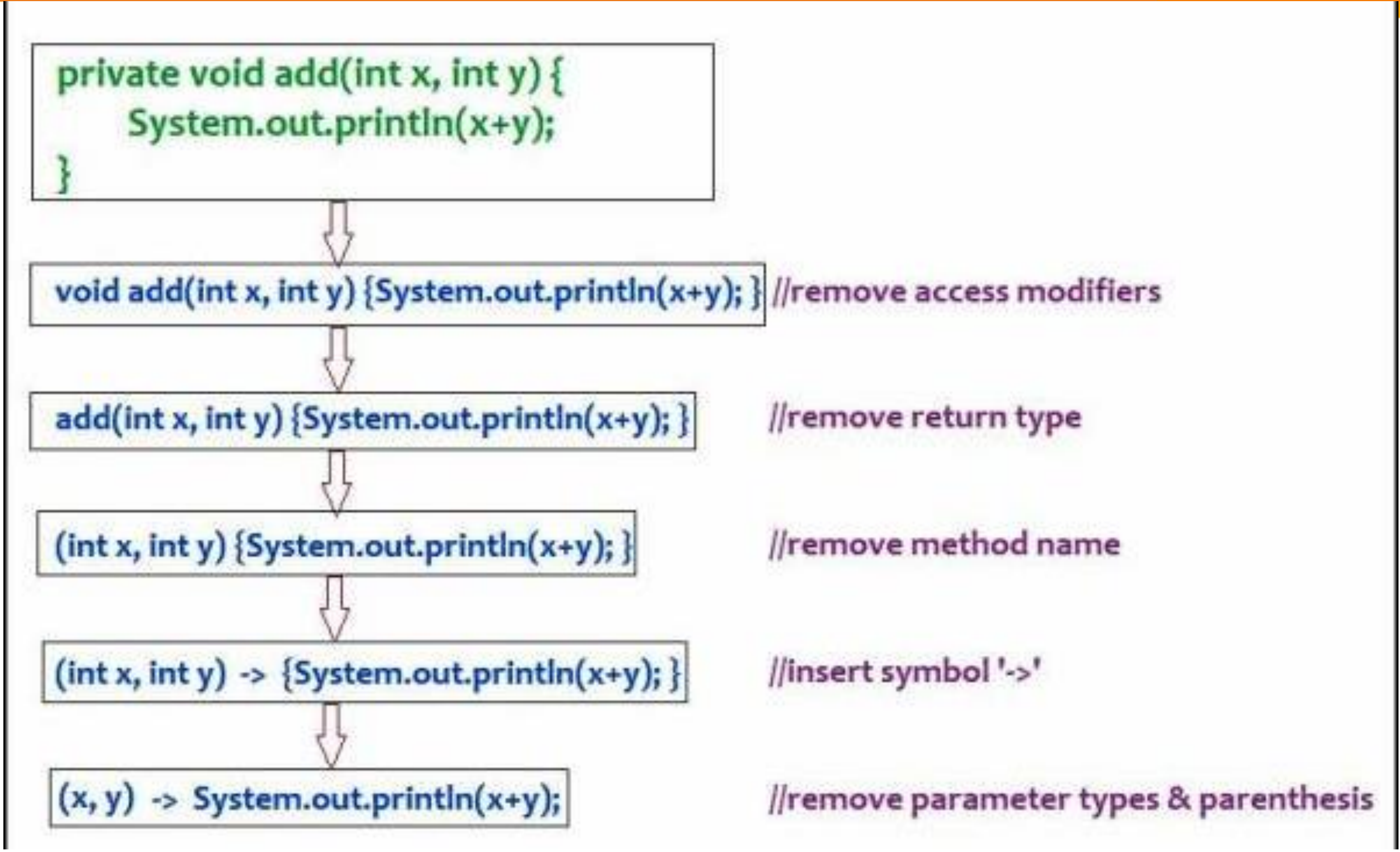
- Lambda Expressions were added in Java 8. A lambda expression is a **short block of code which takes in parameters and returns a value.** Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

- Lambda expression (or function) is an **anonymous function, i.e., a function with no name and any identifier**.

- Lambda expressions are **nameless functions** given as constant values, and written exactly in the place where it's needed, typically as a parameter to some other function.

- **If an interface contain only one abstract method is known as functional interface.** Functional interface can be used with **@FunctionalInterface** annotation with the functional interface.

Syntax: lambda expression code



```
private void add(int x, int y) {
      System.out.println(x+y);
}
```

```
void add(int x, int y) {System.out.println(x+y); }   //remove access modifiers
```

```
add(int x, int y) {System.out.println(x+y); }        //remove return type
```

```
(int x, int y) {System.out.println(x+y); }           //remove method name
```

```
(int x, int y)  ->  {System.out.println(x+y); }      //insert symbol '->'
```

```
(x, y)  ->  System.out.println(x+y);                 //remove parameter types & parenthesis
```

Syntax: lambda expression

Syntax:

(parameters) -> expression

No parameter Syntax:

() -> {

//Body of no parameter lambda

} ;

One parameter Syntax:

(p1) -> {

//Body of single parameter lambda

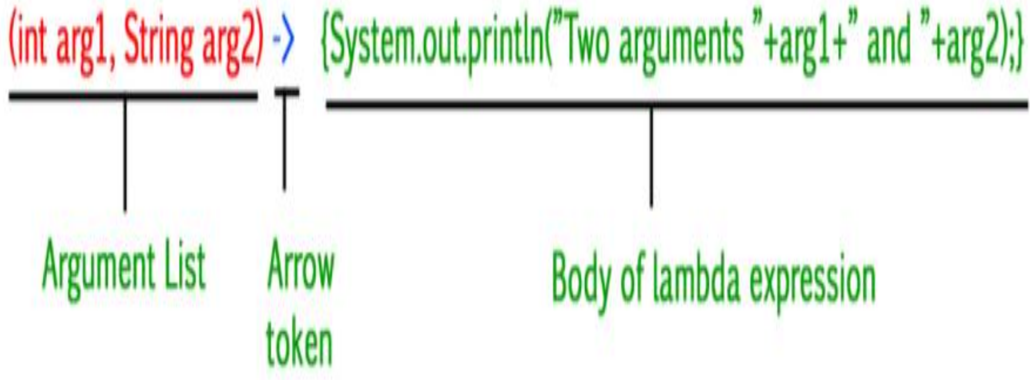
};

Two parameter Syntax:

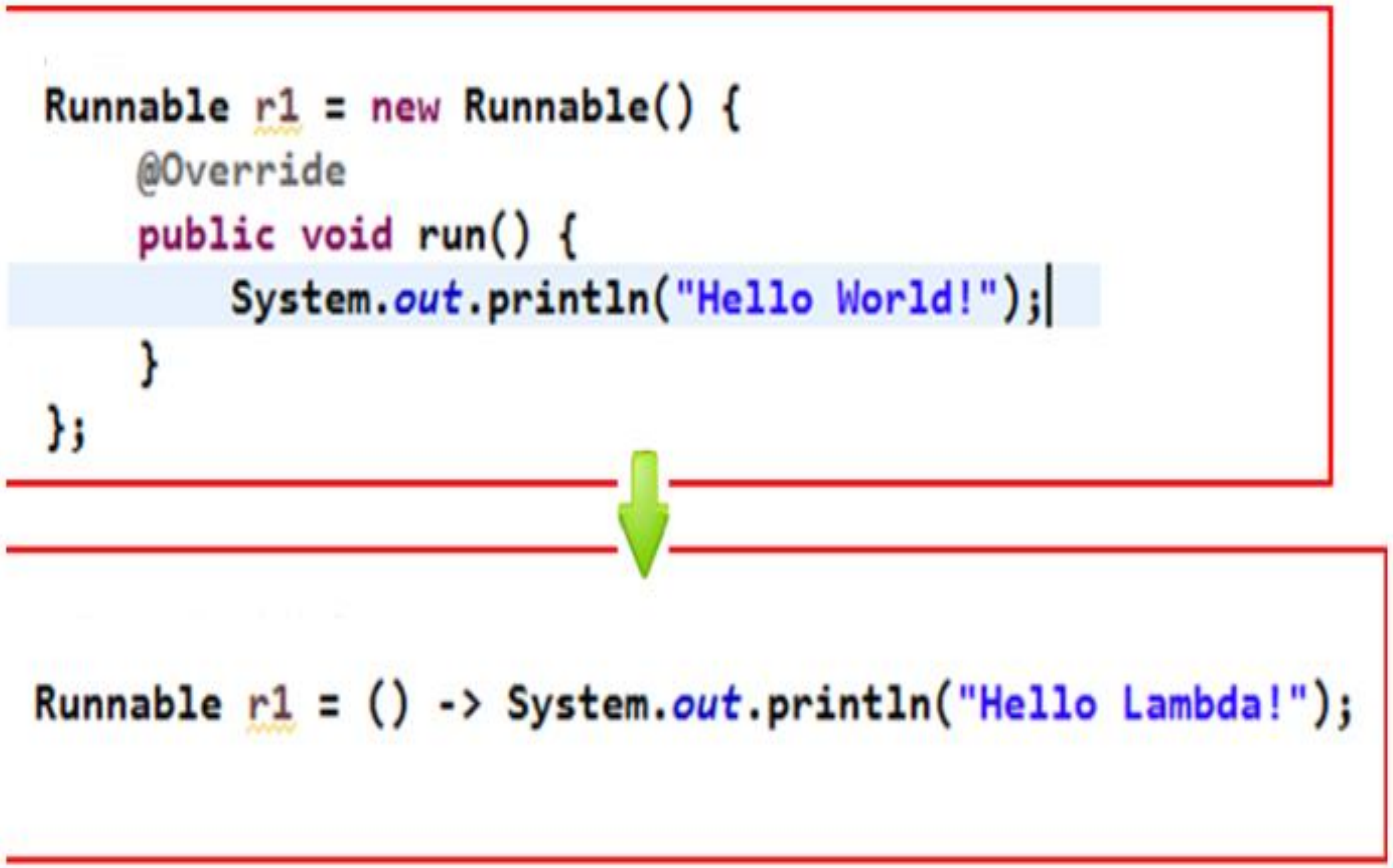
(p1,p2) -> {

//Body of multiple parameter lambda

} ;


(int arg1, String arg2) -> {System.out.println("Two arguments "+arg1+" and "+arg2);}
Argument List
Arrow token
Body of lambda expression

Syntax: lambda expression



```java
Runnable r1 = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello World!");
    }
};
```

```java
Runnable r1 = () -> System.out.println("Hello Lambda!");
```

Need for lambda expression

➢ **Need for Lambda Expression**

It is very useful in **collection library, it helps to iterate, filter and extract data** from collection. It provides below functionalities:

1. Enable to treat functionality as a **method argument, or code as data.**

2. A function that can be **created without belonging to any class**.

3. A lambda expression **can be passed around as if it was an object** and executed on demand.

Example1 of lambda expression (without parameters)

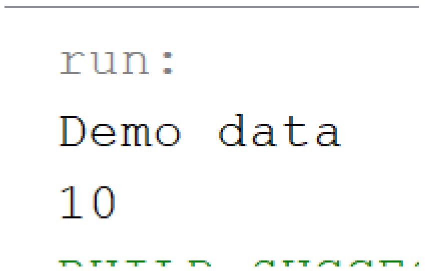
```
package example;
interface Demo{
    public void draw();
}
public class Example {
    public static void main(String[] args) {
            int len=4;
            Demo d2=()->{ System.out.println("Demo data: "+(len*5));  };
        d2.draw();
    }
}
```

Example2 of lambda expression (with parameters)

```
package example;
interface Demo{
    public void draw(int length);
}
public class Example {
    public static void main(String[] args) {
            Demo d2=(len)->{ System.out.println("Demo data: "+(len*5));  };
        d2.draw(10);
    }
}
```

Example3 of lambda expression

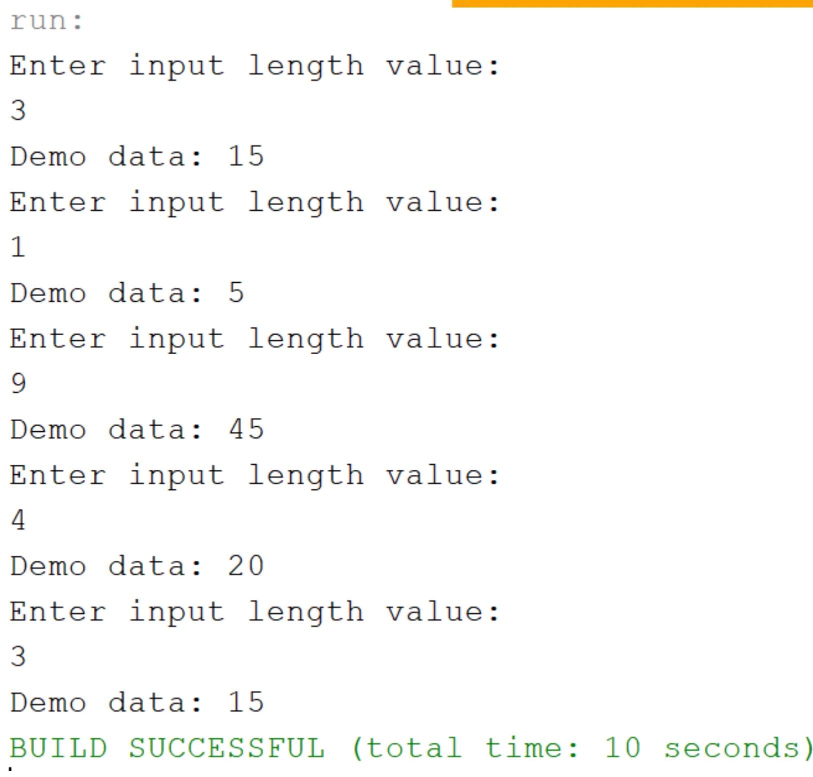
```
package example;
interface Demo{
    public int draw(int length);
}
public class Example {
    public static void main(String[] args) {
            Demo d2=(len)->{ System.out.println("Demo data”);
                                    return len;  };
        System.out.println(d2.draw(10));
    }
}
```



```
run:
Demo data
10
```

Example4 of lambda expression

```
import java.util.*;
//FUNCTIONAL INTERFACE
interface Demo{
   public void absdraw();
}
public class Example {
   public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
    for(int i=0;i<5;i++) {
      System.out.println("Enter input length value: ");
      int len=sc.nextInt();
      Demo d2=()->{ System.out.println("Demo data: "+(len*5)); };
      d2.absdraw(); }
   }
}
```



```
run:
Enter input length value:
3
Demo data: 15
Enter input length value:
1
Demo data: 5
Enter input length value:
9
Demo data: 45
Enter input length value:
4
Demo data: 20
Enter input length value:
3
Demo data: 15
BUILD SUCCESSFUL (total time: 10 seconds)
```

Type Inference

- Type Inference means that **the data type of any expression (eg. method return type or parameter type) can be deduced automatically by the compiler.**
- Type inference is a Java compiler's ability to look at each method invocation and corresponding declaration to determine the type argument (or arguments) that make the invocation applicable.
- The **inference algorithm determines the types of the arguments** and, if available, the type that the result is being assigned, or returned. Finally, the inference algorithm tries to find the most specific type that works with all of the arguments.
- Example: (Older version)

```
List<Integer> list  = new List<Integer>();
```

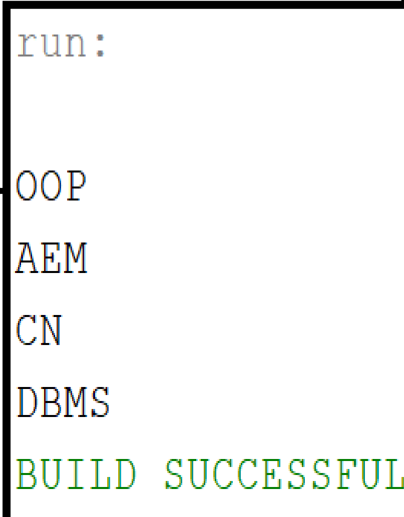
can be written as (Newer version)

```
List<Integer> list = new List<>();
```

Type Inference

```
import java.util.*;


class SubjectName

{

   String sname;


  public SubjectName(String sname)

  {

       this.sname = sname;

    }

}
```

```
public class MyLambdaDemo
{
    public static void main(String args[])
{
        List<SubjectName> subjectList = new ArrayList<>();
         subjectList.add(new SubjectName(""));
         subjectList.add(new SubjectName("OOP"));
         subjectList.add(new SubjectName("AEM"));
         subjectList.add(new SubjectName("CN"));
         subjectList.add(new SubjectName("DBMS"));
         // print using foreach
 subjectList.forEach((subj) ->
System.out.println(subj.sname));
    }
}
```



```
run:


OOP
AEM
CN
DBMS
BUILD SUCCESSFUL
```

Type Inference

```
import java.util.List;
import java.util.ArrayList;
public class TypeInference {
    public static void main(String[] args) {

List<Integer> list = new ArrayList<Integer>();
        list.add(12);
        for (Integer element : list) {
            System.out.println(element);
        }
```

```
        List<Integer> list2 = new ArrayList<>();
        list2.add(12);
        for (Integer element : list2) {
            System.out.println(element);
        }
    }
}
```