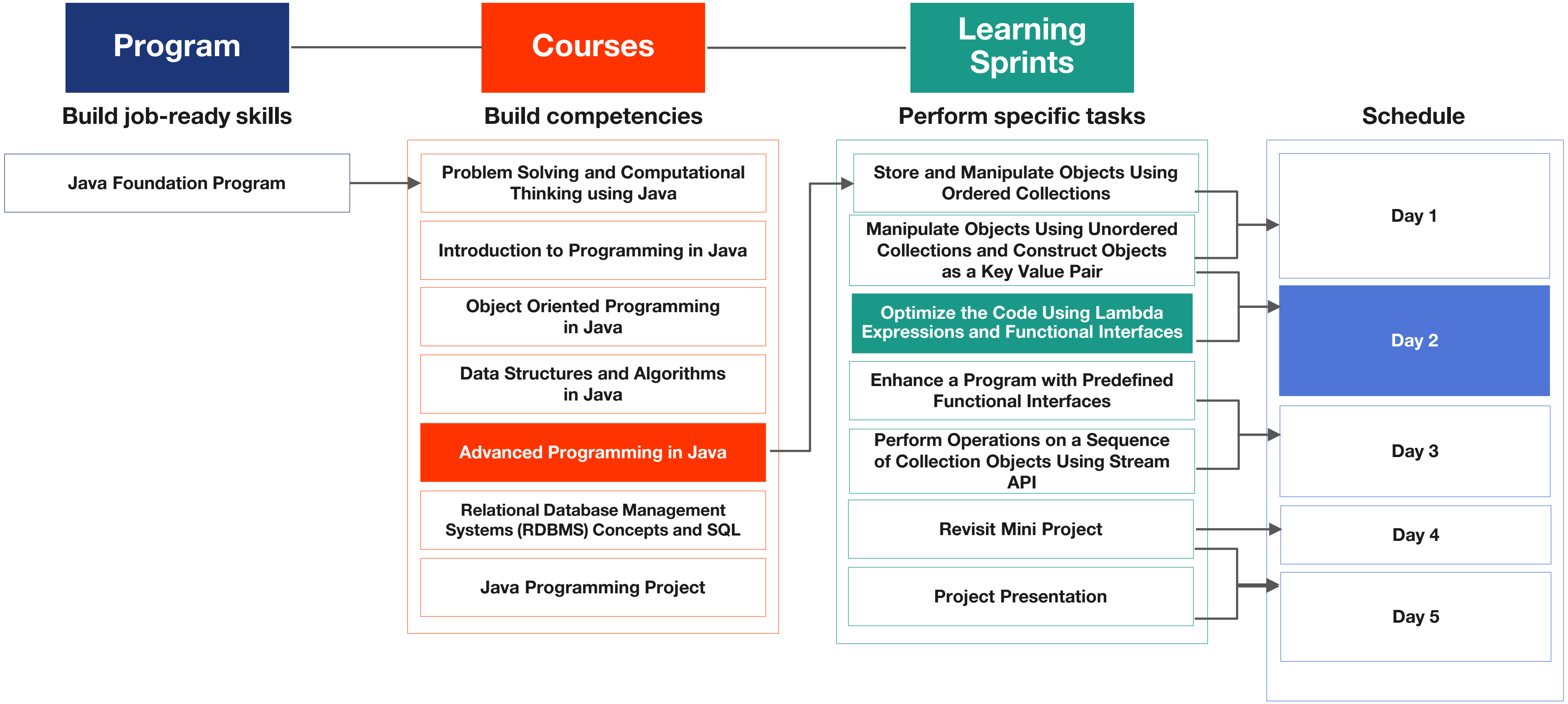
Java Program: Course 5: Plan


Program
Build job-ready skills
Java Foundation Program
Courses
Build competencies
Problem Solving and Computational Thinking using Java
Introduction to Programming in Java
Object Oriented Programming in Java
Data Structures and Algorithms in Java
Advanced Programming in Java
Relational Database Management Systems (RDBMS) Concepts and SQL
Java Programming Project
Learning Sprints
Perform specific tasks
Store and Manipulate Objects Using Ordered Collections
Manipulate Objects Using Unordered Collections and Construct Objects as a Key Value Pair
Optimize the Code Using Lambda Expressions and Functional Interfaces
Enhance a Program with Predefined Functional Interfaces
Perform Operations on a Sequence of Collection Objects Using Stream API
Revisit Mini Project
Project Presentation
Schedule
Day 1
Day 2
Day 3
Day 4
Day 5

Think and Tell

The administration of a particular school needs to perform the following operations on their students' data.

- Sort the names of all the students in alphabetical order
- Sort the students according to the total marks scored by them in descending order
- Find the top three performers

How can we design a program to perform these tasks?

The `NameComparator` Class

```
public class NameComparator
implements Comparator<Student> {

@Override
public int compare(Student o1,
Student o2) {
return
(o1.getStudentName().compareTo(o2
.getStudentName()));
     }
}
```

A `NameComparator` object can be implemented to sort students’ names alphabetically.

The `MarksComparator` Class

A `MarksComparator` object can be implemented to sort students' marks in descending order and to get the top three performers

```
public class MarksComparator
implements
Comparator<Student> {
@Override
public int compare(Student
o1, Student o2) {
 return o2.getTotalMarks() -
o1.getTotalMarks();
    }
}
```

The Main Class

```
List<Student> studentList = new
ArrayList<>();
studentList.add(new Student("Raj", 245));
studentList.add(new Student("Tina", 405));
studentList.add(new Student("Sam", 445));
studentList.add(new Student("Tom", 455));
studentList.add(new Student("Hari", 385));
studentList.add(new Student("Yanni", 485));
studentList.add(new Student("Tim", 345));
studentList.add(new Student("Ria", 405));
studentList.add(new Student("Uma", 345));
studentList.add(new Student("Gary", 405));
studentList.add(new Student("Polly", 345));
studentList.add(new Student("Ravi", 405));
```

```
// Sort names in alphabetical order
Collections.sort(studentList,new
NameComparator());
for (Student s: studentList   ) {
        System.out.println(s);
         }
// Sort in descending order of marks and
// retrieve the first 3 elements
Collections.sort(studentList,new
MarksComparator());
System.out.println("The top three students
are : ");
System.out.println(studentList.get(0));
System.out.println(studentList.get(1));
System.out.println(studentList.get(2));
```

Optimize the Code Using Lambda Expressions and Functional Interfaces

Learning Objectives

- Define lambda expressions
- Implement lambda expression and block lambdas
- Use lambda expressions with functional interfaces

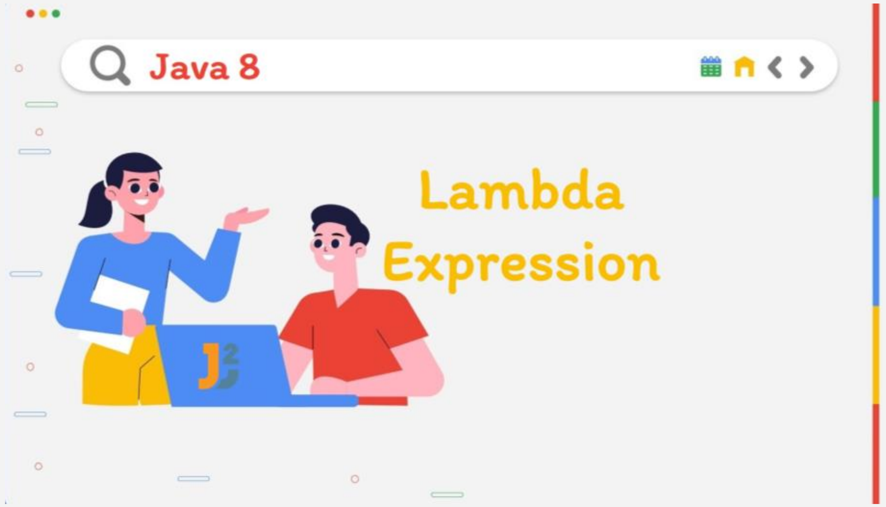
Lambda Expressions

Lambdas are essentially anonymous functions that can be passed to and returned from other functions.

Source: https://commons.wikimedia.org/

Advantages of Lambdas

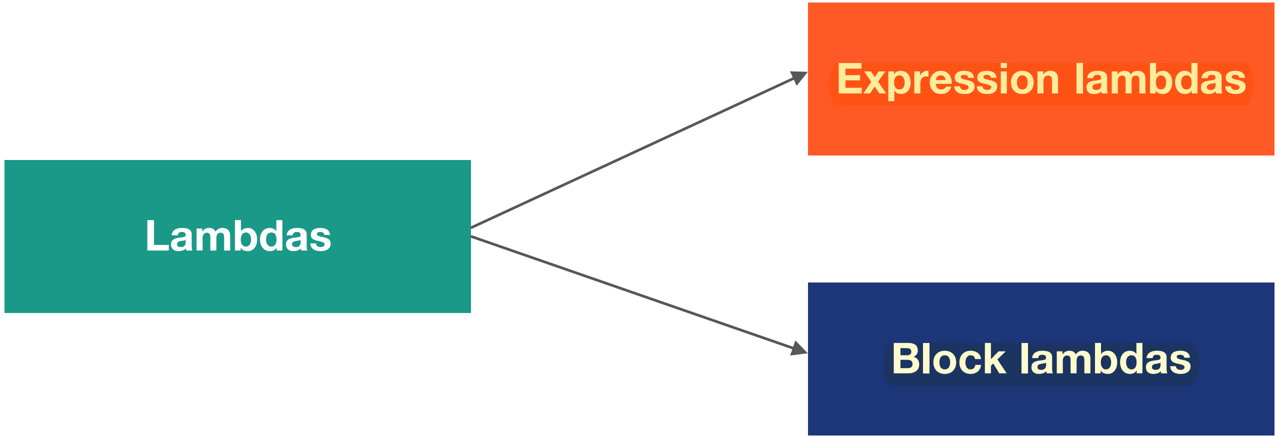
- Concise code
- No repetitive statements
- Readable and effective programs
- Enhanced productivity



Source: https://java2blog.com/

Types of Lambdas

Lambdas are of two types:



Syntax for Expression Lambda

The syntax for writing an expression lambda is:

```
(parameters) -> <expression> ;
```

Expression Lambdas

Sorting a product list by price:

The `Collections.sort` takes two parameters:

- Product list
- Comparator object replaced by an expression lambda

```
List<Product> productList =
Arrays.asList(new Product("Soap",
5),
new Product("Shampoo", 15),
new Product("DishWash Liquid", 8),
new Product("Comb", 5),
new Product("Plastic cup", 5),
new Product("Washing Soap", 12),
new Product("Paper Cups", 19));

Collections.sort(productList, (o1,
o2) -> (int) (o2.price -
o1.price));
```

Syntax for Block Lambda

The syntax for writing a block lambda is:

```
(parameters) -> {statement block} ;
```

Block Lambdas

```
interface Numbers
{
String OddOrEven(int num);
}
// In the main method
Numbers n = num->
{
String s = (num%2==0)?
"Even":"Odd";return s;
};

String str = n.OddOrEven(50);
System.out.println(str);
```

- Let us use lambdas to determine if a number is odd or even

Quick Check!

A lambda expression can have

1. One parameter only
2. Zero parameters
3. Two parameters
4. Zero or more parameters

How Are Interfaces Used in Lambda

- The interface used in a lambda expression must have only one abstract method
- The `Comparator` interface below has only one abstract method; thus, it can be used along with a lambda expression

```
interface Comparator<T> {

     compare(T o1, T o2)

}
```

- This kind of an interface is called a functional interface

User-Defined Functional Interfaces

- Java allows users to write their own functional interfaces
- To ensure that an interface can be used with a lambda, we can specify it with an annotation

```
@FuntionalInterface

interface Addition {

int add(int number1,int number2);

}
```

- Annotation is a marker that tells the compiler that an interface is a functional interface and it can have only one abstract method

Quick Check!

What should be the output of the following code snippet?

```
interface StringConcat {
public String sconcat(String a, String b);
}
public class Example {
public static void main(String args[]) {
StringConcat s = (str1, str2) -> str1 + str2;
System.out.println("Result: "+s.sconcat("Hello", "World"));
}
}
```

School Administration

The administration of a particular school needs to perform the following operations on their students' data.

1. Sort the names of all the students in alphabetical order.
2. Sort the students according to the total marks scored by them in descending order and find the top three performers.

Write a program to perform these tasks using lambda expressions.

Key Takeaways

- Lambda expressions
- Advantages of lambdas
- Types of lambda expressions
- Use of interfaces in lambdas
- User-defined functional interfaces

Thank you!