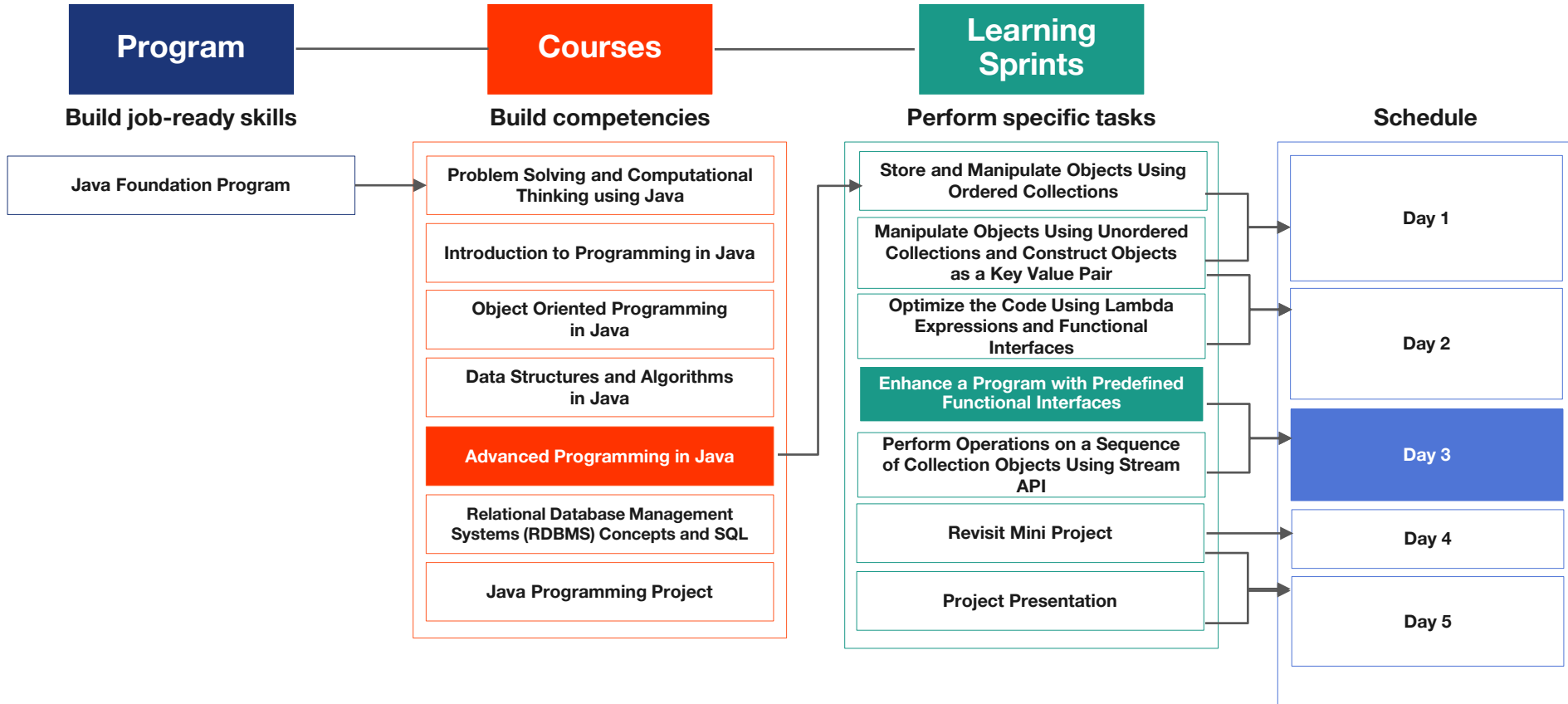


# Java Program: Course 5: Plan



# Think and Tell

We know that we can use programming to solve mathematical problems.

For example, we can

- Find the factorial of a number
- Find the greatest common divisor of a number

How can we use Lambdas to perform such mathematical programming tasks?



# Computing Factorial with Lambda

```
@FunctionalInterface
interface FactorialCalculation
{
    public int factorial(int number);
}

public class Factorial {

    public static void main(String[] args) {
        FactorialCalculation fact = (number) ->
    {
        int result = 1;
        for (int i = 1; i <= number; i++) {
            result = i * result;
        }
        return result;
    };
    System.out.println(fact.factorial(5));
}
```

Let us see how we can find the factorial of a number using lambda expressions.

## Factorial Calculations

Is there a more efficient way to compute the factorial that does not require us to write a functional interface?

Are there any Java 8 libraries providing built-in or predefined interfaces to be used?

```
@FunctionalInterface
interface FactorialCalculation
{
    public int factorial(int number);
}

public class Factorial {

    public static void main(String[] args) {
        FactorialCalculation fact = (number) ->
    {
        int result = 1;
        for (int i = 1; i <= number; i++) {
            result = i * result;
        }
        return result;
    };
    System.out.println(fact.factorial(5));
}
```

# Enhance a Program with Predefined Functional Interfaces



# Learning Objectives

- Describe the usage of Consumer<T>, Supplier<T>, Predicate<T>, and Function<T,R> interfaces
- Use predefined Java 8 functional interfaces



# Functional Interfaces in Java 8

- Java 8 provides predefined functional interfaces to perform functional programming using lambdas
- The `java.util.function` package has all the predefined functional interfaces
- The most commonly used functional interfaces are

Interface	Description
<b>Consumer&lt;T&gt;</b>	Applies an operation on an object of type T. Its method is <b>accept()</b>
<b>Function&lt;T,R&gt;</b>	Applies an operation on an object of type T and returns a result of type R. Its method is <b>apply()</b>
<b>Predicate&lt;T&gt;</b>	Determines if the object of type T fulfills a constraint and returns true or false. Its method is <b>test()</b>
<b>Supplier&lt;T&gt;</b>	Returns an object of type T. Its method is <b>get()</b>

## The Consumer<T> Interface

The Consumer<T> interface has one abstract method

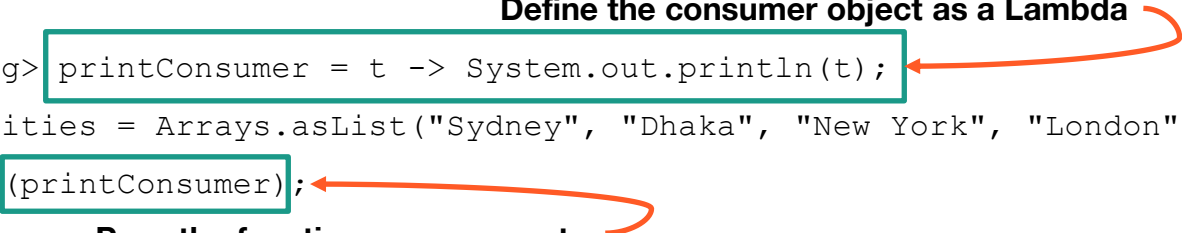
```
void accept(T t) ;
```

- Consumer<T> is a Java 8 built-in functional interface that accepts a single input and returns no output



# Usage – The Consumer<T> Interface

```
public void printAllCityNames()
{
    Define the consumer object as a Lambda
    Consumer<String> printConsumer = t -> System.out.println(t);
    List<String> cities = Arrays.asList("Sydney", "Dhaka", "New York", "London");
    cities.forEach(printConsumer);
    Pass the function as a parameter
}
```



- The `forEach` is an internal `for` loop used to iterate over the objects of any collection.
- It takes an object of the `Consumer` interface as a parameter and helps iterate through the collection
- A lambda function is passed as a parameter for the `forEach` loop

## The Supplier<T> Interface

- The Supplier<T> interface represents a supplier of results
- Supplier<T> does the opposite of Consumer<T>
- It does not take any argument and only returns some value

The Supplier<T> interface has one method

```
T get()
```

# Usage – The Supplier<T> Interface

The `randomNumbers` object of the `Supplier` interface supplies a random number to any class between 0 and 10

```
Supplier<Integer> randomNumbers = () -> new Random().nextInt(10);
```

# Quick Check!

To replace the question mark (?) in the code snippet given below, which option would you select to print the list elements?

```
list.forEach(  
    (name) -> System.out.println(?)  
);
```

1. “ “
2. 'name'
3. 'list'



## The Predicate<T> Interface

The Predicate<T> interface has one abstract method

```
boolean test(T t)
```

- Predicate<T> is a generic functional interface
- It represents a single argument function that returns a boolean value (true or false)
- It is mainly used to filter data

# Usage – The Predicate<T> Interface

**Define the filter method**

```
static void filter(List<Person> list, Predicate<Person> pre)
{
    for(Person l:list)
    {
        if(pre.test(l))
            System.out.println(l);
    }
}
```

**In the main method, use the predicate to get all names starting with 'C'**

```
Predicate<Person> per = p->p.getFname().startsWith("C");
filter(plist,per);
```

# The Function $\langle T, R \rangle$ Interface

- The Function  $\langle T, R \rangle$  interface is a generic interface that takes one argument and produces a result
- The input type and the return type are specified in the generic interface

The Function $\langle T, R \rangle$  interface has one abstract method

**`apply(T t)`**

# Usage – The Function<T,R> Interface

Function<Integer, Integer> factorial = (n)->{  The function takes an integer as input and returns an integer as output

```
int result = 1;
for(int i = 1; i <= n ;i++)
{
    result = i * result;
}
return result;
};
```

```
System.out.println("The factorial of number 5 is "+factorial.apply(5));
```



# Quick Check!

**Do you think the below code will compile?**

```
@FunctionalInterface
public interface Function2<T, U, V> {
    public V apply(T t, U u);
}
```



## Rewarding Customers on Points Earned

A grocery store needs to analyze the data of its registered customers in order to provide them award points based on the purchases they have made in a month.

Perform the following tasks using predefined functional interfaces.

1. Display the details of the registered customers
2. Find the customers who belong to the specified city.
3. Assume that the zip code of the city has been changed to 3456.  
Accordingly, change the address of all the customers.
4. Calculate the reward points scored by each customer.  
Give 1 reward point for every \$50 spent at the store.



## Key Takeaways

- Predefined functional interfaces in Java 8
- `Consumer<T>`
- `Supplier<T>`
- `Predicate<T>`
- `Function<T,R>`





Thank you!