
*Presents*

# Functional Programming

# Programming Paradigms

- There exist different programming paradigms
  - ✓ These represent different approaches to writing code
  - ✓ Each approach has advantages and disadvantages
  - ✓ Two main styles
- Imperative programming
  - ✓ Code is a series of commands that describe how to get a result
  - ✓ Object oriented and procedural programming are imperative
- Declarative programming
  - ✓ Code describes what the outcomes should be
  - ✓ Functional programming is declarative

# Java Programming

- Java originally was pure OO programming
  - ✓ Functionality is encapsulated in class methods
  - ✓ Methods are written in procedural style
  - ✓ Code is executed by asking an object to execute a method
- Java 8 introduced functional programming constructs
  - ✓ Streams and other features were introduced
  - ✓ Nowadays Java can be used for OO or functional programming
- Most modern programming languages support functional programming
  - ✓ JavScript, Python, Go, etc.

# Functional Programming

- The basic programming unit is a function
  - ✓ Based on the mathematical concept of a function
  - ✓ e.g. Square function: $f(x) = x * x$
  - ✓ Functions do not change the input data
  - ✓ The output of f(x) is a new piece of data created from the input data
- We create complex operations by combining functions
  - ✓ Eg. f(g(x)) produces an output where f() takes as input the result of applying g() to an input x.
  - ✓ Algorithms are expressed as a series of functions representing the steps of the algorithms

LearnQuest

# Functional Programming

- Functions are first class objects in FP
  - ✓ They can be assigned to variables
  - ✓ They can be passes as arguments to other functions
    - This is the f(g(x)) composition
  - ✓ They can be returned from a function
- We create complex operations by combining functions
  - ✓ Eg. f(g(x)) produces an output where f() takes as input the result of applying g() to an input x.
  - ✓ Algorithms are expressed as a series of functions representing the steps of the algorithms

LearnQuest

# Function Variables

- Java variables must have a type
- A function type is:
  - The number and type of arguments it takes
  - It's return value
  - Essentially the function signature plus return type
- We will look at three types of function variables
  - Functions of one argument - functions
  - Functions of two arguments - bifunctions
  - Functions of one argument that returns a Boolean – predicates
  - There are other types available in java.util.function

# Function Variables

- The example below show three function variables

```
import java.util.function.BiFunction;
import java.util.function.Function;
import java.util.function.Predicate;

public class Runner {
    public static Function<Integer,Integer> f;
    public static BiFunction<Integer,Integer,Integer> b;
    public static Predicate<Integer> p;

    public static void main(String[] args) {

    }
}
```

# Imperative OO Programming

- To define a function to do exponentiation with integers
  - ✓ Need to create a place holder class with static methods
  - ✓ There is only one thing we can do with this logic – call it as a method
  - ✓ A method is *not* a first class object
- This is exactly the role of a function in OO programming
  - ✓ Methods are bound to a class
  - ✓ They represent behaviors associated with that type

LearnQuest

# Imperative OO Programming

```
class MyFunctions {
    static int power(int base, int exponent) {
        if (exponent <= 0 ) return 1;
        int p = base;
        for (int i = 1; i < exponent; i++) {
            p = p * base;
        }
        return p;
    }
}

public class Runner {

    public static void main(String[] args) {
        System.out.println(MyFunctions.power(3, 3));

    }
```

# The Function Construct

- A Function<T,R>
  - ✓ Takes an input of type T and returns a result of type R
  - ✓ The function body is now data that can be stored in a variable, data that just happens to be executable
  - ✓ The code has a memory location just like any other variable
  - ✓ The function body is written as a Lambda function
  - ✓ Notice that we are using generics to implement this
  - ✓ To execute the code, we need to sent it an apply message with the appropriate arguments
  - ✓ The function body can be assigned to a different variable of the same type.

# Function Example

```
6  public class Runner {
7
8      public static Function<Integer,Integer> f = x -> x + 1;
9      public static Function<Integer,Integer> g;
10
11     public static void main(String[] args) {
12         System.out.println(f.apply(4) + " Address of f " + f);
13         f = x -> x * x;
14         System.out.println(f.apply(4) + " Address of f " + f);
15         g = f;
16         System.out.println(g.apply(4) + " Address of g " + g);
17
18         };
```

Problems @ Javadoc Declaration Console

```
<terminated> Runner (2) [Java Application] C:\tools\java\Java18\bin\javaw.exe (Jul. 8, 2022, 9:06:28 p.m. – 9:06:28 p.m.) [pid: 16680]
5 Address of f function.Runner$$Lambda$1/0x0000000800c009f0@8efb846
16 Address of f function.Runner$$Lambda$2/0x0000000800c00c18@a09ee92
16 Address of g function.Runner$$Lambda$2/0x0000000800c00c18@a09ee92
```

# The BiFunction Construct

- A Function<T1,T2, R>
  - ✓ Takes input of type T1 and  T2 and returns a result of type R

```java
package bifunction;

import java.util.function.BiFunction;

public class Runner {

    public static BiFunction<Integer,Integer,Integer> bi;
    public static void main(String[] args) {
        bi = (x,y) -> x * y;
        System.out.println(bi.apply(2,3) + " Address of bi " + bi);

    }
```

Problems  @ Javadoc  Declaration  Console

```
<terminated> Runner (3) [Java Application] C:\tools\java\Java18\bin\javaw.exe  (Jul. 8, 2022, 9:19:37 p.m. – 9:19:37 p.m.) [pid: 21092]
6 Address of bi bifunction.Runner$$Lambda$1/0x0000000800c009f0@8efb846
```

# The Predicate Construct

- A Predicate<T>
  - ✓ Takes input of type T and returns a Boolean, the result of some test
  - ✓ Instead of “applying” a predicate, we “test” with it

```
3  import java.util.function.Predicate;
4
5  public class Runner {
6      public static Predicate<Integer> isEven;
7
8      public static void main(String[] args) {
9          isEven = x -> 0 == x % 2 ;
10          System.out.println(isEven.test(13));
11     }
12
```

Problems @ Javadoc Declaration Console

```
<terminated> Runner (4) [Java Application] C:\tools\java\Java18\bin\javaw.exe (Jul. 8, 2022, 9:31:58 p.m. – 9:31:
false
```

# Function Composition

- In math we can combine functions
  - ✓ If f(x) and g(x) are functions then f(g(x)) is a composition
  - ✓ The output of g(x) becomes the input of the function f(x)

- We can do the same in Java
  - We have to ensure the data types match up

```
public static void main(String[] args) {
    boolean test = isEven.test(square.apply(3));
    System.out.println(test);

}
static Function<Integer,Integer> square = x → x + x;
static Predicate<Integer> isEven = x → (0 == ( x % 2));
```

# Anonymous Functions

- Another form of functional programming is the use of anonymous function
  - ✓ Functions are treated as first class objects
  - ✓ Seen earlier as lambda or anonymous function
  - ✓ An autonomous chunk of code
  - ✓ Pure functions always returns the same value for the same arguments and has no side effects
- Pure functional versus functional style
  - ✓ Pure functional programming does not allow any mutability
  - ✓ Functional style allows first class functional objects but allows mutability

# Anonymous Functions

- Recall the previously we implemented a thread using the Runnable interface
  - The problem is that we had to create a class just to act as a container for a run() method.

```
public class Runner2 {

    public static void main(String[] args) {
        Runnable r = () -> System.out.println("      Running in " + Thread.currentThread());
        Thread t = new Thread(r);
        t.start();
    }
```

```
oblems  @ Javadoc  Declaration  Console ×
nated> Runner2 [Java Application] C:\tools\java\Java18\bin\javaw.exe (Jul. 8, 2022, 9:47:33 p.m. – 9:47:33 p.m.) [pid: 22336]
Running in Thread[Thread-0,5,main]
```

# Runnable Functions

- But we are only interested in the run() method
  - ✓ We convert it to a Lambda function
  - ✓ Then have it, on its own, implement the Runnable interface

```java
3  public class Runner2 {
4
5      public static void main(String[] args) {
6          Runnable r = () -> System.out.println("Running Thread");
7          Thread t = new Thread(r);
8          t.start();
9
```

Problems | Javadoc | Declaration | Console | Coverage

```
<terminated> Runner2 [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Mar. 1, 2022, 8:28:40 p.m. – 8:28:42 p.m.)
Running Thread
```

# Runnable Functions

- Even more efficient
  - ✓ The Lambda function can be inferred to be runnable by virtue of being a function

```java
public class Runner2 {

    public static void main(String[] args) {
        //Runnable r = () -> System.out.println("Running Thread");
        Thread t = new Thread(() -> System.out.println("Running Thread"));
        t.start();

```

Problems | Javadoc | Declaration | Console | Coverage

```
<terminated> Runner2 [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Mar. 1, 2022, 8:30:47 p.m. – 8:30:49 p.m.)
Running Thread
```

# Runnable Functions

- Even more efficiently

```
5  public class Runner2 {
6
7      public static void main(String[] args) {
8
9          new Thread(() → System.out.println("Running Anon Thread")).start();
10     }
```

Problems @ Javadoc Declaration Console Coverage

```
<terminated> Runner2 [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Mar. 1, 2022, 8:36:08 p.m. – 8:36:08 p.m.)
Running Anon Thread
```

# Functions as Parameters

```java
public class Runner {
     // function takes a function and a numeric argument
    static Function<Integer,Integer> square = x -> x * x;
    static Function<Integer,Integer> cube = x -> x * x * x;

    static int f(Function<Integer,Integer> func, int arg ) {
        return func.apply(arg);
    }
    public static void main(String[] args) {
        System.out.println("    Squaring " + f(square,4));
        System.out.println("    Cubing " + f(cube,4));
        System.out.println("    Adding one " + f(x -> x + 1,4));

    }
```

blems @ Javadoc Declaration Console

nated> Runner (6) [Java Application] C:\tools\java\Java18\bin\javaw.exe (Jul. 8, 2022, 10:13:30 p.m. – 10:13:30 p.m.) [pid: 13216]

```
Squaring 16
Cubing 64
Adding one 5
```

# Functions as Return Values

```java
public class Runner {
    // Predicate function
    static Predicate<Integer> isEven = x -> 0 == x %2;

    // Static method that returns a predicate function
    static Predicate<Integer> f()  { return isEven;}

    public static void main(String[] args) {
        Predicate<Integer> func = null;
        func = f();
        System.out.println("     Even test for 5 = " + func.test(5));
```

oblems @ Javadoc Declaration Console

nated> Runner (7) [Java Application] C:\tools\java\Java18\bin\javaw.exe (Jul. 9, 2022, 8:38:15 a.m. – 8:38:15 a.m.) [pid: 14216]

```
Even test for 5 = false
```

# Questions