Functional Programming

Bulit-in Functional Interfaces

java.util.function package

- this packages contains many built-in functional interfaces
- in order to use them, you have to know
  - name of the interface
  - signature of the abstract method
  - return type of the abstract method
- this leads to a paradigm usually referred to as *functional programming*

Most Common Functional Interfaces

Functional Interface	Method Signature	Return Type
`Supplier<T>`	`get()`	`T`
`Consumer<T>`	`accept(T)`	`void`
`BiConsumer<T, U>`	`accept(T, U)`	`void`
`Predicate<T>`	`test(T)`	`boolean`
`BiPredicate<T, U>`	`test(T, U)`	`boolean`
`Function<T, R>`	`apply(T)`	`R`
`BiFunction<T, U, R>`	`apply(T, U)`	`R`
`UnaryOperator<T>`	`apply(T)`	`T`
`BinaryOperator<T>`	`apply(T, T)`	`T`

Supplier

```
@FunctionalInterface
public interface Supplier<T> {
  T get();
}
```

```
// supplier example
import java.util.function.*;
import java.time.*;
public class MyClass {
  public static void main(String[] args) {
    Supplier<LocalDateTime> dtImpl = () -> LocalDateTime.now();
    System.out.println(dtImpl.get());
  }
}
```

implementing get() method

```
=> prints out current local date and time
```

Consumer, BiConsumer

```java
@FunctionalInterface
public interface Consumer<T> {
  void accept(T t);
  // ...
}

@FunctionalInterface
public interface BiConsumer<T, U> {
  void accept(T t, U u);
  // ...
}
```

```java
// consumer example

import java.util.function.*;

public class MyClass {

  public static void main(String[] args) {

    Consumer<String> greet = s -> System.out.println("Hello, " + s + "!");

    greet.accept("John Wayne");


    BiConsumer<String, Integer> p =

     (name, age) -> System.out.println(name + " is " + age + " years old.");

    p.accept("John", 40);

  }

}
```

implementing accept(T t) method

implementing accept(T t, U u) method

```
Hello, John Wayne!

John is 40 years old.
```

Predicate, BiPredicate

```
@FunctionalInterface
public interface Predicate<T> {
  boolean test(T t);
  // ...
}

@FunctionalInterface
public interface BiPredicate<T, U> {
  boolean test(T t, U u);
  // ...
}
```

```
// predicate example

import java.util.function.*;

public class MyClass {

  public static void main(String[] args) {

    Predicate<Integer> gt10 = n -> n > 10;

    System.out.println(gt10.test(7) + " " + gt10.test(12));


    BiPredicate<Integer, Integer> gt = (n, m) -> n > m;

    System.out.println(gt.test(7, -1) + " " + gt.test(-7, 1));

  }

}
```

implementing `test(T t)` method

implementing `test(T t, U u)` method

```
false true

true false
```

Function, BiFunction

```
@FunctionalInterface
public interface Function<T, R> {
  R apply(T t);
  // ...
}

@FunctionalInterface
public interface BiFunction<T, U, R> {
  R apply(T t, U u);
  // ...
}
```

```java
// function example
import java.util.function.*;
public class MyClass {
  public static void main(String[] args) {
```

implementing apply(T t) method

```java
    Function<Integer, Double> square = n -> (double)(n*n);
    var res = square.apply(5);
    System.out.println(res);
```

implementing apply(T t, U u) method

```java
    BiFunction<String, Integer, String> con = (s, i) -> s + i;
    var myCon = con.apply("John", 25);
    System.out.println(myCon);
  }
}
```

```
25.0
John25
```

UnaryOperator, BinaryOperator

```
@FunctionalInterface
public interface UnaryOperator<T> extends Function<T, T> {
  // ...
}


@FunctionalInterface
public interface BinaryOperator<T> extends BiFunction<T, T, T> {
  // ...
}
```

```java
public class MyClass {
  public static void main(String[] args) {
```

implementing apply(T t) method

```java
    UnaryOperator<Integer> negative = n -> -n;
    System.out.println(negative.apply(5));
```

implementing apply(T t) method

```java
    UnaryOperator<String> shout = String::toUpperCase;
    System.out.println(shout.apply("John"));
```

implementing apply(T t, T u) method

```java
    BinaryOperator<Double> add = (a, b) -> a + b;
    System.out.println(add.apply(3.5, 1.5));
```

implementing apply(T t, T u) method

```java
    BinaryOperator<String> con = String::concat;
    System.out.println(con.apply("John", "Wayne"));
  }
}
```

```
-5
JOHN
5.0
JohnWayne
```