Lecture 66 AutoBoxing and Type Paramerter:

Autoboxing:

**package** com.durgaSoft.section7.lecture66;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

//Autoboxing:

/\*

\* The below code wont work until java 1.4.

\* After 1.5 it works due to Auto Boxing

\* where the below code gets converted to

\* Integer.valueOf(10)

\* \*/

Integer number1 = 10 ;

}

}

Autoboxing

**package** com.durgaSoft.section7.lecture66;

**public** **class** Example2 {

**public** **static** **void** main(String[] args) {

//AutoUnboxing:

/\*

\* The below code wont work until java 1.4.

\* After 1.5 it works due to AutoUnBoxing

\* where the below code gets converted

\* from Wrapper Object to primitive variable

\* \*/

Integer I = **new** Integer(10);

**int** x =I;

}

}

GenericType Parameter:

**import** java.util.ArrayList;

**public** **class** Example3 {

**public** **static** **void** main(String[] args) {

ArrayList<Integer> numbers = **new** ArrayList<Integer>();

}

}

We can only pass Integer object to this ArrayList.

Lecture67: Primitive Type Functional Interface.

Why primitive Type Functional Interface are needed?

Consider the below example. We have an integer array and we need to print numbers that are even. This can be realized as shown below.

**package** com.durgaSoft.section7.lecture67;

**import** java.util.function.Predicate;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

**int**[] integerNumbers = {0,5,10,15,20,25};

Predicate<Integer> checkEvenNumber = x -> x%2==0;

**for** (**int** i : integerNumbers) {

**if**(checkEvenNumber.test(i)) {

System.***out***.println(i);

}

}

}

}

This above approach will have performance issues as we need to convert from primitive int to wrapper object regularly.

The same problem is with other Predefined Functional interfaces. Consider the below example where we are going to perform the square of a number.

**package** com.durgaSoft.section7.lecture67;

**import** java.util.function.Function;

**public** **class** Example2 {

**public** **static** **void** main(String[] args) {

**int**[] integerNumbers = {0,5,10,20,25};

Function<Integer, Integer> square = number -> number \*number;

**for** (**int** i : integerNumbers) {

System.***out***.println(square.apply(i));

}

}

}

To over come these we can use IntPredicate so that there is no AutoBoxing and AutoUnBoxing. Here the difference between these two are we are using IntPredicate instead of Predicate due to which there is no autoBoxing and Auto Unboxing.

**package** com.durgaSoft.section7.lecture68;

**import** java.util.function.IntPredicate;

**import** java.util.function.Predicate;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

**int**[] integerNumbers = {0,5,10,15,20,25};

IntPredicate checkEvenNumber = x -> x%2==0;

**for** (**int** i : integerNumbers) {

**if**(checkEvenNumber.test(i)) {

System.***out***.println(i);

}

}

}

}

For Predicate the similar interfaces which are available are LongPredicate and DoublePredicate.

Lecture 69: Demo Programs for PrimiteType Functional Interfaces for Function.

Program to perform square of a number.

**package** com.durgaSoft.section7.lecture67;

**import** java.util.function.Function;

**public** **class** Example2 {

**public** **static** **void** main(String[] args) {

**int**[] integerNumbers = {0,5,10,20,25};

Function<Integer, Integer> square = number -> number \*number;

**for** (**int** i : integerNumbers) {

System.***out***.println(square.apply(i));

}

}

}

By using IntFunction:

Note: we are returning Integer type but the return type is still an Object.

**package** com.durgaSoft.section7.lecture69;

**import** java.util.function.Function;

**import** java.util.function.IntFunction;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

**int**[] integerNumbers = { 0, 5, 10, 20, 25 };

IntFunction<Integer> square = number -> number \* number;

**for** (**int** i : integerNumbers) {

System.***out***.println(square.apply(i));

}

}

}

Another Example: Where we are using ToIntFunction.

**import** java.util.function.Function;

**import** java.util.function.IntFunction;

**import** java.util.function.ToIntFunction;

**public** **class** Example2 {

**public** **static** **void** main(String[] args) {

Function<String, Integer> getLength1 = string -> string.length();

System.***out***.println(getLength1.apply("Prashanth"));

ToIntFunction<String> getLength2 = string -> string.length();

System.***out***.println(getLength2.applyAsInt("Prashanth Angadikunnath"));

}

}

Example:

**package** com.durgaSoft.section7.lecture69;

**import** java.util.function.Function;

**import** java.util.function.IntToDoubleFunction;

**public** **class** Example3 {

**public** **static** **void** main(String[] args) {

// To calculate SquareRoot of a Number

Function<Integer, Double> squareRoot1 = number -> Math.*sqrt*(number);

System.***out***.println(squareRoot1.apply(4));

System.***out***.println(squareRoot1.apply(5));

IntToDoubleFunction squareRoot2 = number -> Math.*sqrt*(number);

System.***out***.println(squareRoot2.applyAsDouble(4));

System.***out***.println(squareRoot2.applyAsDouble(5));

}

}

One is primitive type and the other is Object type Consumer:

When we have one type as Object and the other as int, long or double then we can use the below interfaces respectively.

ObjIntConsumer, ObjLongConsumer, ObjDoubleConsumer.

Let us Revisit our old example where we are trying to increment salary of Employee.

**package** com.durgaSoft.section7.lecture71;

**import** java.util.function.BiConsumer;

**import** java.util.function.ObjDoubleConsumer;

**public** **class** Example2 {

**public** **static** **void** main(String[] args) {

//old example

BiConsumer<Employee, Double> empIncrement1 = (emp,increment)-> emp.salary=emp.salary+increment;

ObjDoubleConsumer<Employee> empIncrement2 = (emp,increment)-> emp.salary=emp.salary+increment;

Employee employee1 =**new** Employee("Employee1", 1000.0);

Employee employee2 =**new** Employee("Employee2", 500.0);

System.***out***.println("Employee before increment "+employee1);

System.***out***.println("Employee before increment "+employee2);

empIncrement1.accept(employee1, 100.0);

empIncrement2.accept(employee2, 10);

System.***out***.println("Employee after increment "+employee1);

System.***out***.println("Employee after increment "+employee2);

System.***out***.println();

}

}

**class** Employee{

String name;

**double** salary;

@Override

**public** String toString() {

**return** "Name=" + name + ", Salary=" + salary;

}

**public** Employee(String name, **double** salary) {

**super**();

**this**.name = name;

**this**.salary = salary;

}

}

Lecture 72: Primitive version for Supplier:

**package** com.durgaSoft.section7.lecture72;

**import** java.util.Random;

**import** java.util.function.IntSupplier;

**import** java.util.function.Supplier;

/\*

\* To generate 6 digit OTP

\*

\* Logic

\* Random r = new Random();

\* int low = 10;

\* int high = 100;

\* int result = r.nextInt(high-low) + low;

\*

\* This gives you a random number in between 10 (inclusive) and 100 (exclusive)

\* \*\*/

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

Random random = **new** Random();

Supplier<Integer> randomNumber1 = () -> random.nextInt(1000000-100000)+100000;

System.***out***.println(randomNumber1.get());

IntSupplier randomNumber2 = () -> random.nextInt(1000000-100000)+100000;

System.***out***.println(randomNumber2.getAsInt());

}

}

Lecture 73: Unary Operator

Consider this example where we want to get square of an integer.

**package** com.durgaSoft.section7.lecture73;

**import** java.util.function.Function;

**import** java.util.function.UnaryOperator;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

/\*

\* Below both the input and return type are always of the same type then

\* we need to go for Unary Operator

\*/

Function<Integer, Integer> squareOfANumber1 = number1 -> number1\* number1;

System.***out***.println(squareOfANumber1.apply(5));

/\*In Unary Operator we specify only a single type it will take both for

\* Input and Return type.

\*/

UnaryOperator<Integer> squareOfANumber2 = number1 -> number1\* number1;

System.***out***.println(squareOfANumber2.apply(5));

}

}

There is still a problem in the above example we are still using Integer Wrapper Object. If you want int or Double Long you can use IntUnaryOperator or DoubleUnaryOperator or LongUnaryOperator.

The above example can we re written as:

**package** com.durgaSoft.section7.lecture73;

**import** java.util.function.Function;

**import** java.util.function.IntUnaryOperator;

**import** java.util.function.UnaryOperator;

**public** **class** Example2 {

**public** **static** **void** main(String[] args) {

IntUnaryOperator squareOfANumber2 = number1 -> number1\* number1;

System.***out***.println(squareOfANumber2.applyAsInt(2));

}

}

Lecture 74: Binary Operator:

Function -> when we have one input and returns one Output

BiFunction -> where we have two input and returns one Output

BinaryOperator -> Similar to BiFunction but Where all the input and return type is are the same.

Here we are going to concatenate two Strings.

**package** com.durgaSoft.section7.lecture74;

**import** java.util.function.BiFunction;

**import** java.util.function.BinaryOperator;

**public** **class** Example {

**public** **static** **void** main(String[] args) {

BiFunction<String, String, String> concatinateMessag1 = (message1, message2) -> message1+" "+message2;

System.***out***.println(concatinateMessag1.apply("Prashanth ", "Angadikunnath"));

BinaryOperator<String> concatinateMessage2 = (message1, message2) -> message1+" "+message2;

System.***out***.println(concatinateMessage2.apply("Prashanth ", "Angadikunnath"));

}

}

Lecture 75: Primitive Version of Binary Operator:

For int, long and double we are having IntBinaryOperator, LongBinaryOperator and DoubleBinaryOperator

To Sum of BinaryOperator

**package** com.durgaSoft.section7.lecture75;

**import** java.util.function.BinaryOperator;

**import** java.util.function.IntBinaryOperator;

**public** **class** Example1 {

**public** **static** **void** main(String[] args) {

BinaryOperator<Integer> sumOf2Number1 =(num1, num2) -> num1+num2;

System.***out***.println(sumOf2Number1.apply(2, 3));

IntBinaryOperator sumOf2Number2 =(num1, num2) -> num1+num2;

System.***out***.println(sumOf2Number2.applyAsInt(2, 3));

}

}