Introduction to Java Programming

Introduction to the new Java 8 Features Available

Java Basics

- Students should be able to understand:
  - StringBuffer
  - StringJoiner
  - Functional Interfaces
  - default and static methods in Interfaces
  - Generic Functional Interfaces

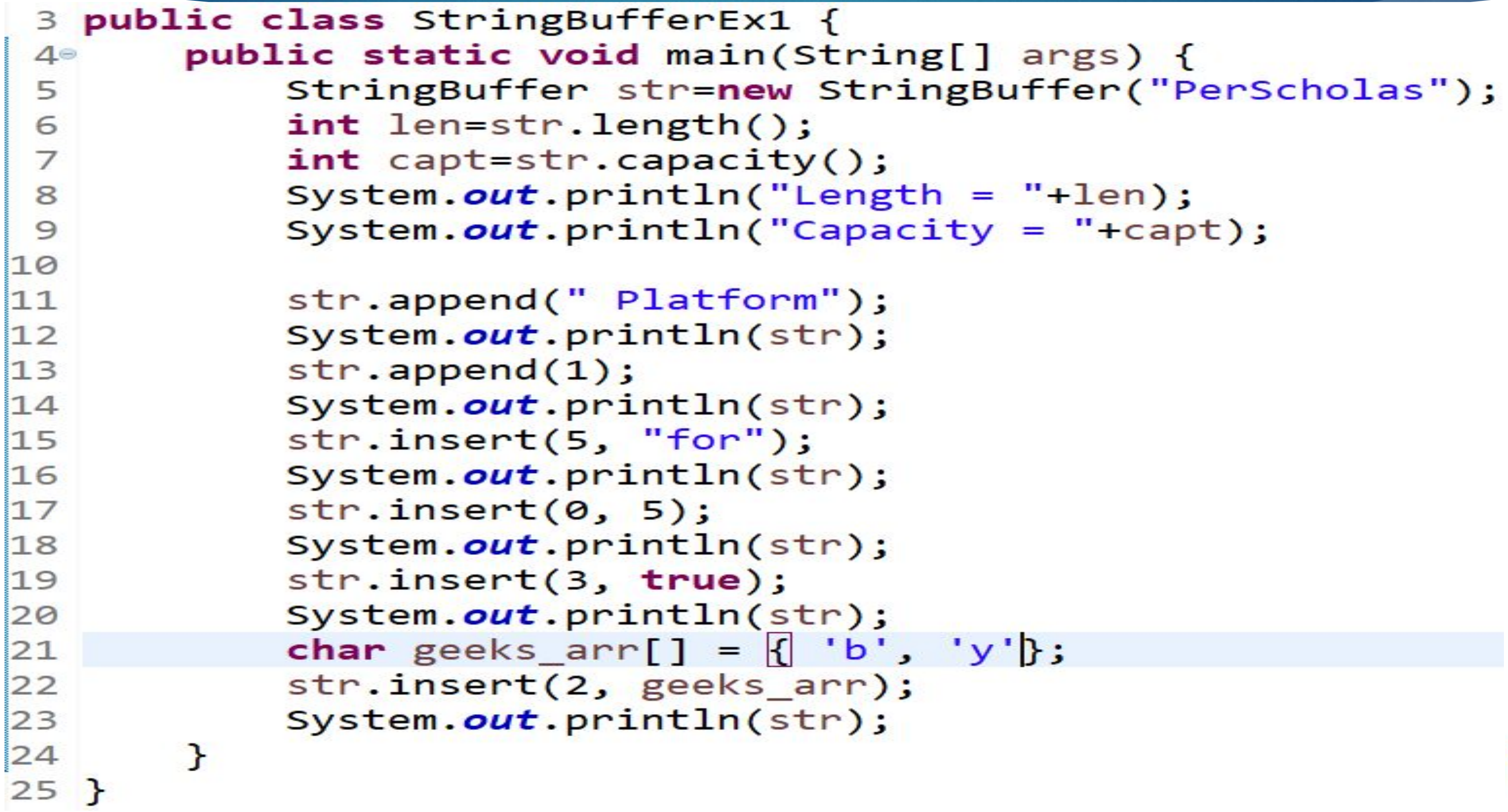
Java Basics

StringBuffer

- StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.
- StringBuffer may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.
- StringBuffer Constructors
  - StringBuffer( ): It reserves room for 16 characters without reallocation.
    - **StringBuffer s=new StringBuffer();**
  - StringBuffer(int size)It accepts an integer argument that explicitly sets the size of the buffer.
    - **StringBuffer s=new StringBuffer(20);**
  - StringBuffer(String str): It accepts a String argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.
    - **StringBuffer s=new StringBuffer("PerScholas");**

Java Basics

StringBuffer

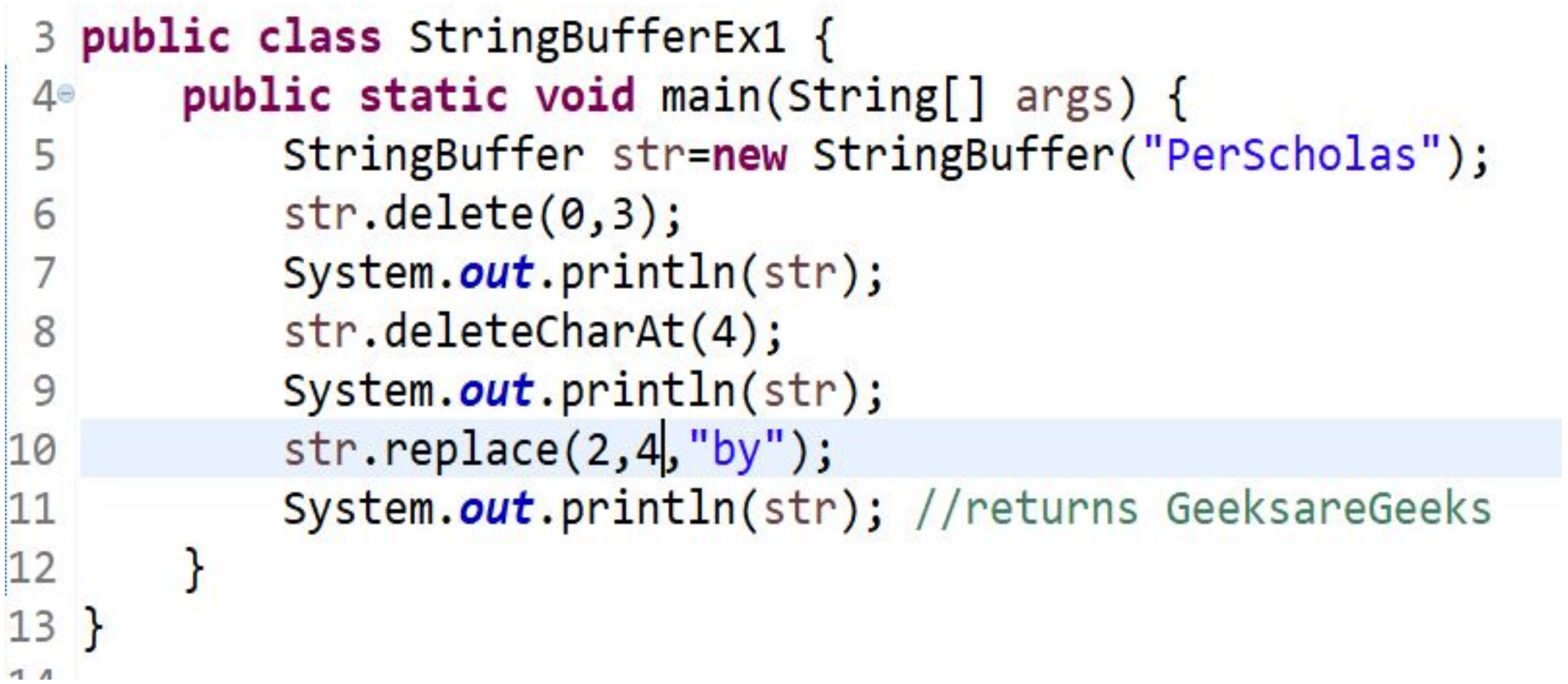


```
public class StringBufferEx1 {
    public static void main(String[] args) {
        StringBuffer str=new StringBuffer("PerScholas");
        int len=str.length();
        int capt=str.capacity();
        System.out.println("Length = "+len);
        System.out.println("Capacity = "+capt);

        str.append(" Platform");
        System.out.println(str);
        str.append(1);
        System.out.println(str);
        str.insert(5, "for");
        System.out.println(str);
        str.insert(0, 5);
        System.out.println(str);
        str.insert(3, true);
        System.out.println(str);
        char geeks_arr[] = { 'b', 'y'};
        str.insert(2, geeks_arr);
        System.out.println(str);
    }
}
```

Java Basics

StringBuffer



```
public class StringBufferEx1 {
    public static void main(String[] args) {
        StringBuffer str=new StringBuffer("PerScholas");
        str.delete(0,3);
        System.out.println(str);
        str.deleteCharAt(4);
        System.out.println(str);
        str.replace(2,4,"by");
        System.out.println(str); //returns GeeksareGeeks
    }
}
```

Java Basics

StringJoiner

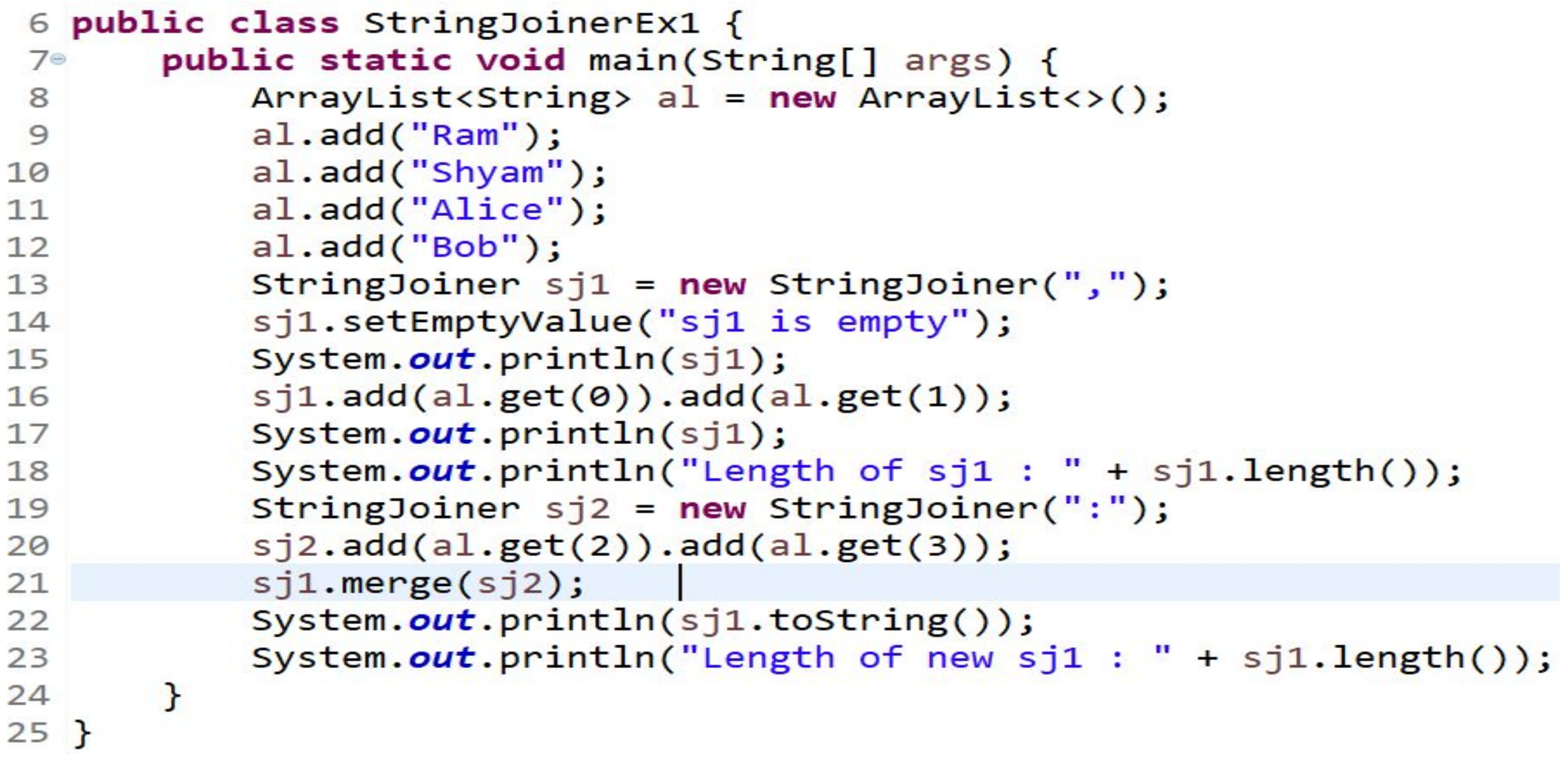
- StringJoiner is a class in java.util package which is used to construct a sequence of characters(strings) separated by a delimiter and optionally starting with a supplied prefix and ending with a supplied suffix. Though this can also be with the help of StringBuilder class to append delimiter after each string, but StringJoiner provides easy way to do that without much code to write.

Java Basics

StringJoiner

Constructors :

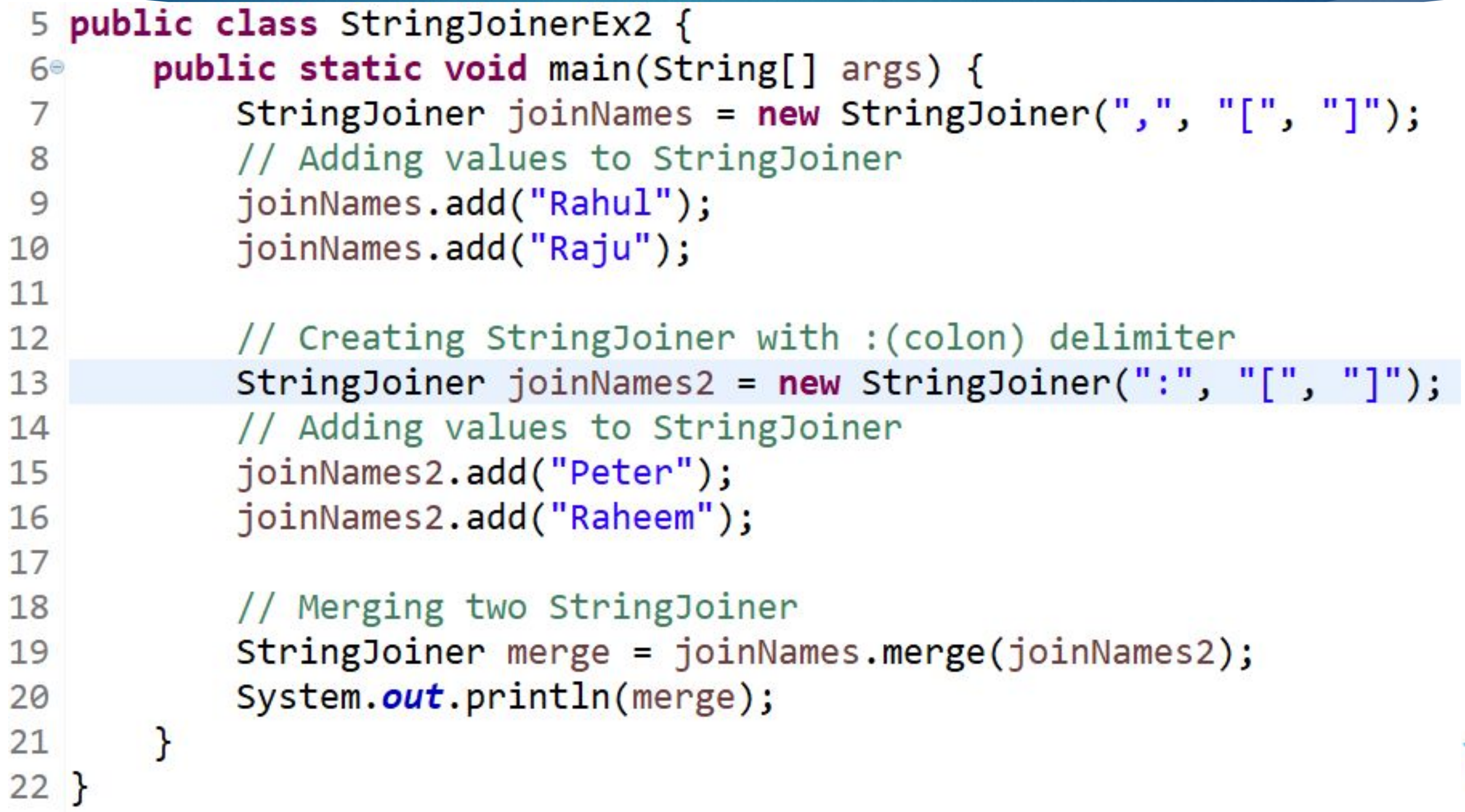
StringJoiner(CharSequence delimiter) : Constructs a StringJoiner with no characters in it, with no prefix or suffix, and a copy of the supplied delimiter.



```java
public class StringJoinerEx1 {
    public static void main(String[] args) {
        ArrayList<String> al = new ArrayList<>();
        al.add("Ram");
        al.add("Shyam");
        al.add("Alice");
        al.add("Bob");
        StringJoiner sj1 = new StringJoiner(",");
        sj1.setEmptyValue("sj1 is empty");
        System.out.println(sj1);
        sj1.add(al.get(0)).add(al.get(1));
        System.out.println(sj1);
        System.out.println("Length of sj1 : " + sj1.length());
        StringJoiner sj2 = new StringJoiner(":");
        sj2.add(al.get(2)).add(al.get(3));
        sj1.merge(sj2);
        System.out.println(sj1.toString());
        System.out.println("Length of new sj1 : " + sj1.length());
    }
}
```

Java Basics

StringJoiner



```java
public class StringJoinerEx2 {
    public static void main(String[] args) {
        StringJoiner joinNames = new StringJoiner(",", "[", "]");
        // Adding values to StringJoiner
        joinNames.add("Rahul");
        joinNames.add("Raju");

        // Creating StringJoiner with :(colon) delimiter
        StringJoiner joinNames2 = new StringJoiner(":", "[", "]");
        // Adding values to StringJoiner
        joinNames2.add("Peter");
        joinNames2.add("Raheem");

        // Merging two StringJoiner
        StringJoiner merge = joinNames.merge(joinNames2);
        System.out.println(merge);
    }
}
```

Java Basics

Functional Interface

- In java 8 context, functional interface is an interface having exactly one abstract method called functional method to which the lambda expression's parameter and return types are matched. Functional interface provides target types for lambda expressions and method references.
- The java.util.function contains general purpose functional interfaces used by JDK and also available for end users like us.
- The interfaces defined in the this package are annotated with FunctionalInterface. This annotation is not the requirement for the java compiler to determine the interface is an functional interface but it helps the compiler to identify the accidental violation of the design intent.

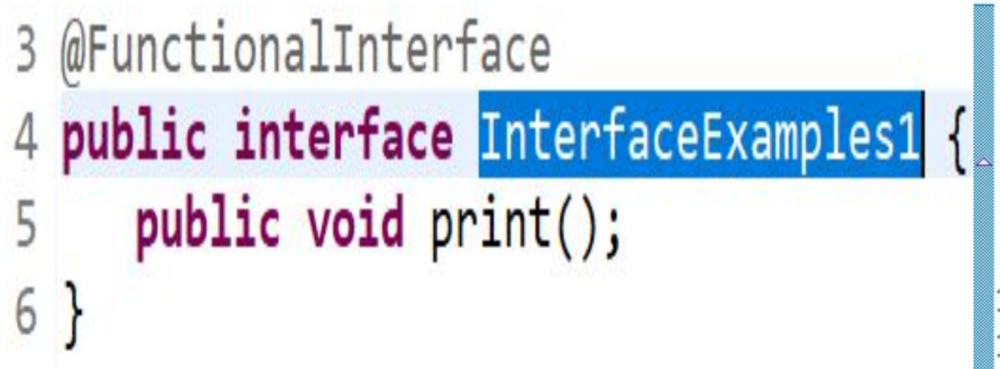
Java Basics

@FunctionalInterface Annotation

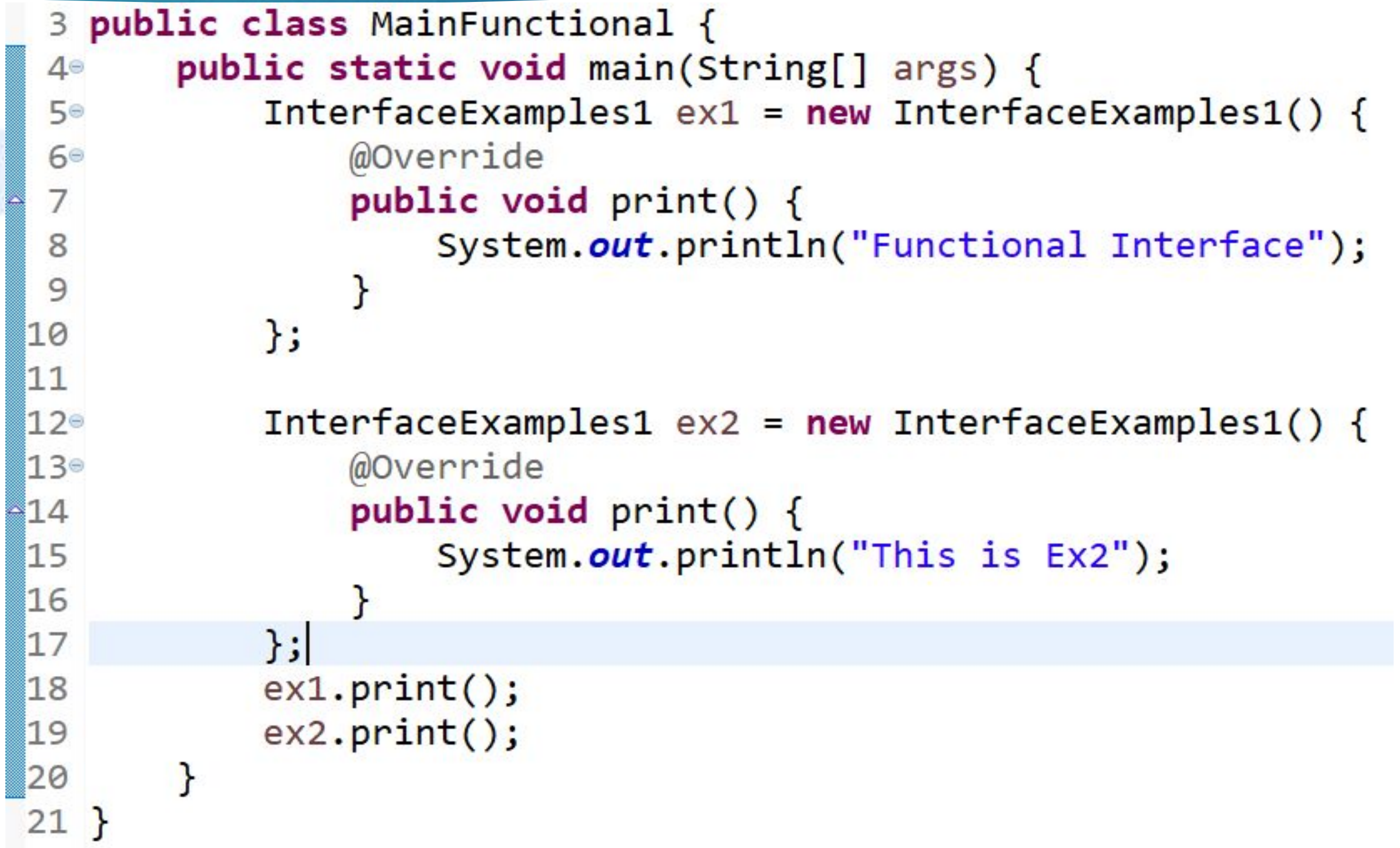
- As discussed @FunctionalInterface is a runtime annotation that is used to verify the interface follows all of the rules that can make this interface as functional interface.
- @FunctionalInterface Rules:
  - Interface must have exactly one abstract method.
  - It can have any number of default methods because they are not abstract and implementation is already provided by same.
  - Interface can declares an abstract method overriding one of the public method from java.lang.Object, that still can be considered as functional interface. The reason is any implementation class to this interface will have implementation for this abstract method either from super class (bare minimum java.lang.Object) class or defined by implementation class it self. In the below example toString() method declared as abstract which will be implemented in its concrete implementation class or at last derived from java.lang.Objectclass.

Java Basics

@FunctionalInterface Annotation



```java
@FunctionalInterface
public interface InterfaceExamples1 {
    public void print();
}
```



```java
public class MainFunctional {
    public static void main(String[] args) {
        InterfaceExamples1 ex1 = new InterfaceExamples1() {
            @Override
            public void print() {
                System.out.println("Functional Interface");
            }
        };

        InterfaceExamples1 ex2 = new InterfaceExamples1() {
            @Override
            public void print() {
                System.out.println("This is Ex2");
            }
        };
        ex1.print();
        ex2.print();
    }
}
```

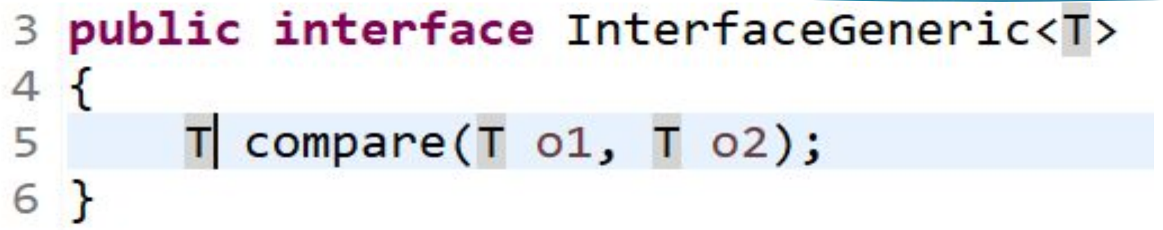
Java Basics

Generic Functional Interface

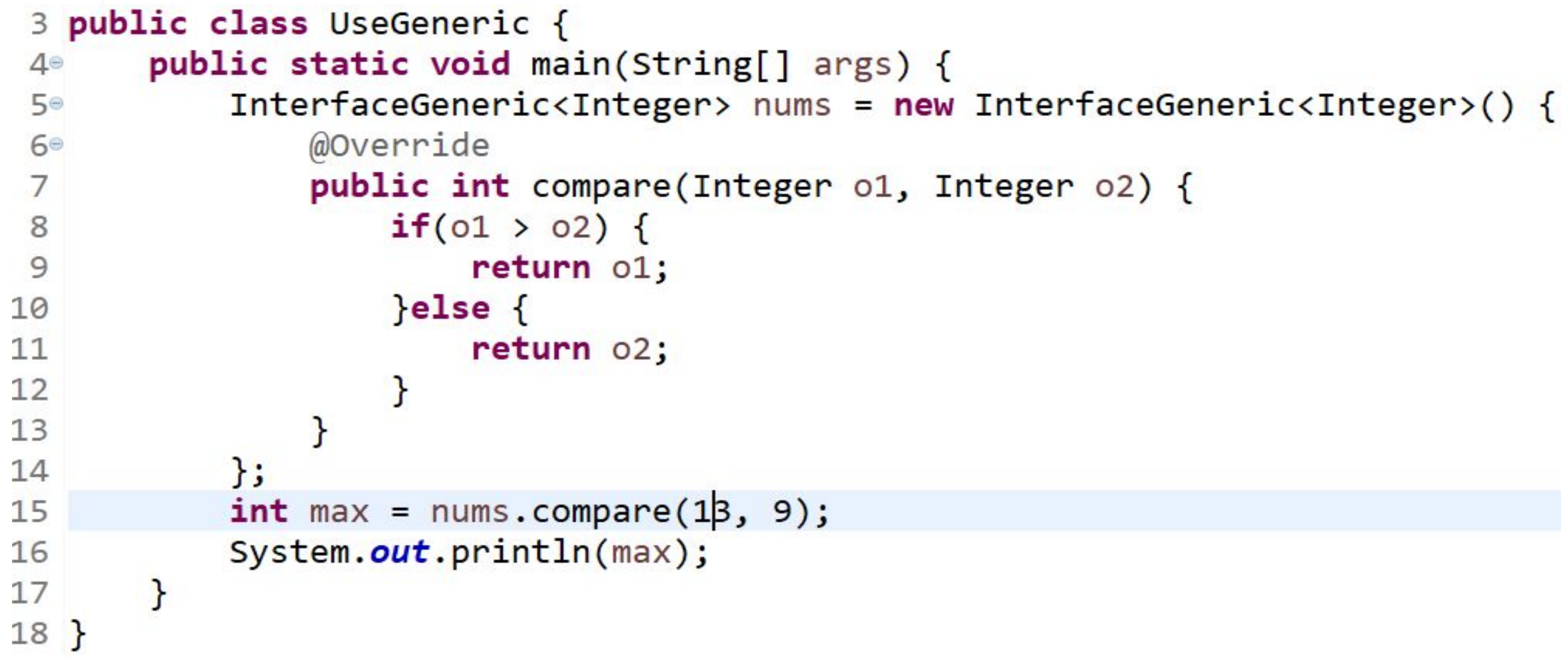
- Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.
- Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.
- We can use type parameters with a functional interface to create generic functional interface.

Java Basics

Generic Functional Interface



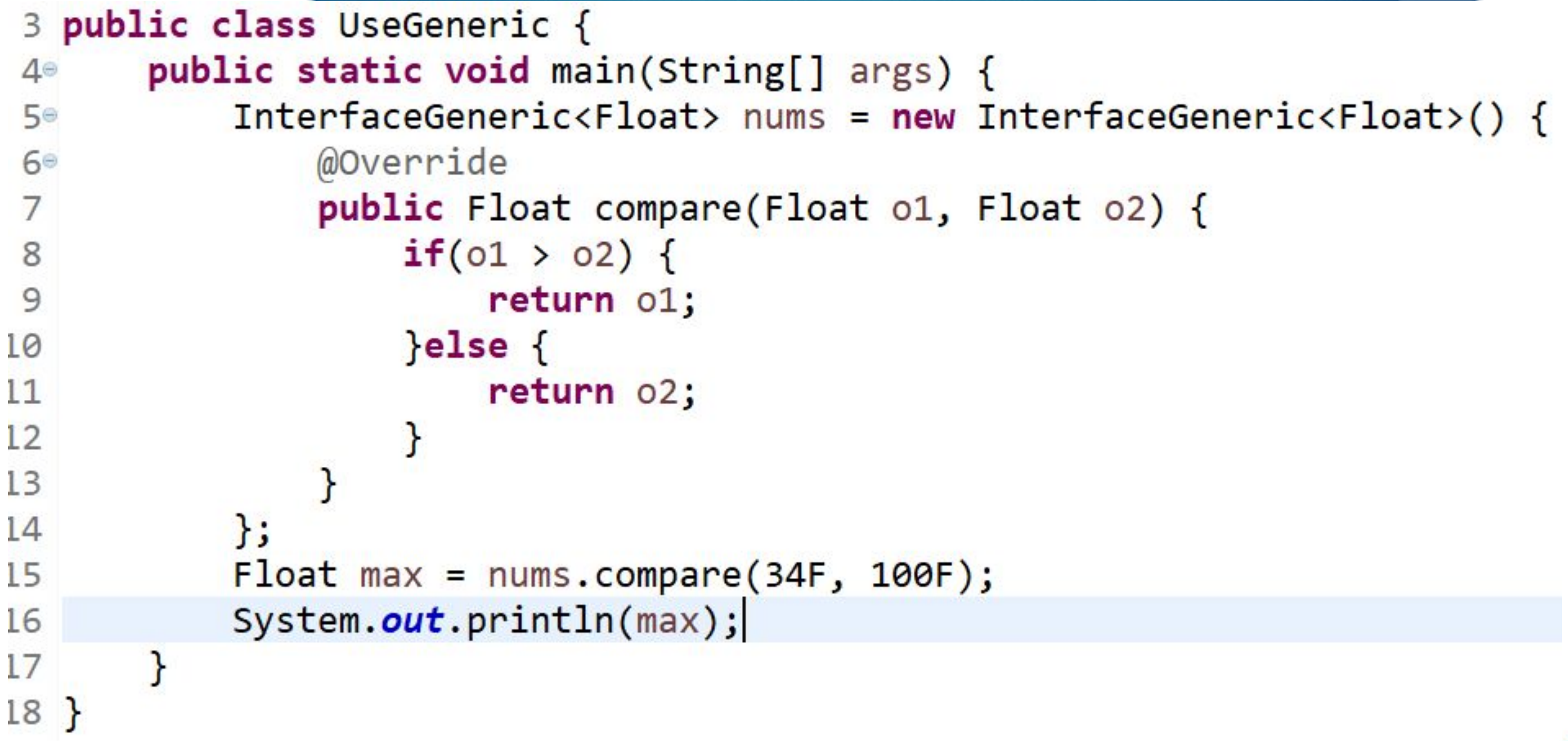
```java
public interface InterfaceGeneric<T>
{
	T compare(T o1, T o2);
}
```



```java
public class UseGeneric {
    public static void main(String[] args) {
        InterfaceGeneric<Integer> nums = new InterfaceGeneric<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                if(o1 > o2) {
                    return o1;
                }else {
                    return o2;
                }
            }
        };
        int max = nums.compare(13, 9);
        System.out.println(max);
    }
}
```

Java Basics

Generic Functional Interface



```java
public class UseGeneric {
    public static void main(String[] args) {
        InterfaceGeneric<Float> nums = new InterfaceGeneric<Float>() {
            @Override
            public Float compare(Float o1, Float o2) {
                if(o1 > o2) {
                    return o1;
                }else {
                    return o2;
                }
            }
        };
        Float max = nums.compare(34F, 100F);
        System.out.println(max);
    }
}
```

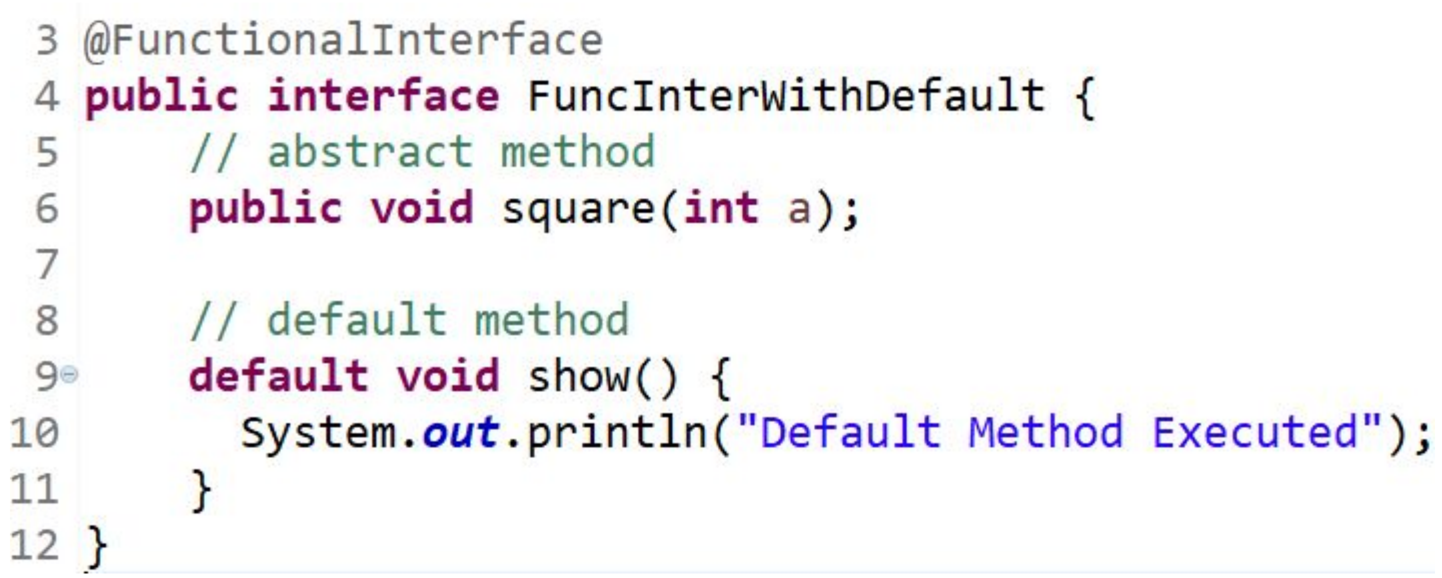
Java Basics

Default Methods

Before Java 8, interfaces could have only abstract methods. The implementation of these methods has to be provided in a separate class. So, if a new method is to be added in an interface then its implementation code has to be provided in the class implementing the same interface. To overcome this issue, Java 8 has introduced the concept of default methods which allow the interfaces to have methods with implementation without affecting the classes that implement the interface.

Java Basics

Default Methods

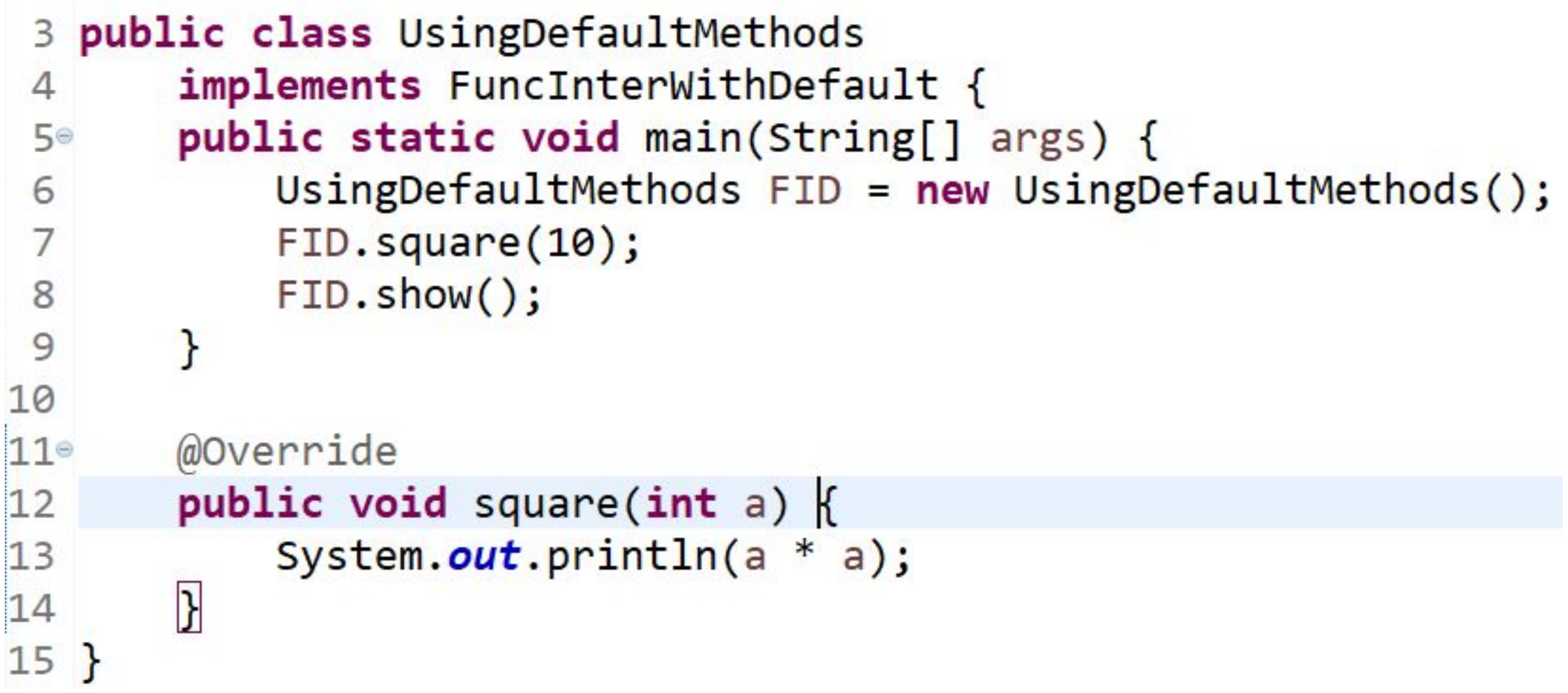


```
@FunctionalInterface
public interface FuncInterWithDefault {
    // abstract method
    public void square(int a);

    // default method
    default void show() {
      System.out.println("Default Method Executed");
    }
}
```

Java Basics

Default Methods



```java
public class UsingDefaultMethods
    implements FuncInterWithDefault {
    public static void main(String[] args) {
        UsingDefaultMethods FID = new UsingDefaultMethods();
        FID.square(10);
        FID.show();
    }

    @Override
    public void square(int a) {
        System.out.println(a * a);
    }
}
```

Java Basics

Simple Calculator

- ❑ Create a simple calculator that can add two numbers:
- ❑ Create an Interface Calc that has a single method called: compute
  - ❑ This method takes an array of integers
- ❑ Create a class called: MainEntry. In this class implement the compute method four times by using blocks to add, subtract, multiply and divide. Then ask the user to enter a series of numbers and what they should like to do with those numbers. In case of divide, only allow two numbers