**Java**

***Package, Interface &***

***Exception***

***Package***

Prepared By - Rifat Shahriyar 2

Package

* Java package provides a mechanism for partitioning the class name space into more manageable chunks

– Both **naming** and **visibility** control mechanism

* Define classes inside a package that are not accessible by code outside that package
* Define class members that are exposed only to other members of the same package
* This allows classes to have intimate knowledge of each other

– Not expose that knowledge to the rest of the world

Prepared By - Rifat Shahriyar 3

Declaring Package

* ***package pkg***

– Here, pkg is the name of the package

* ***package MyPackage***

– creates a package called MyPackage

* The package statement defines a name space in which classes are stored
* If you omit the package statement, the class names are put into the **default package**, which has no name

Prepared By - Rifat Shahriyar 4

Declaring Package

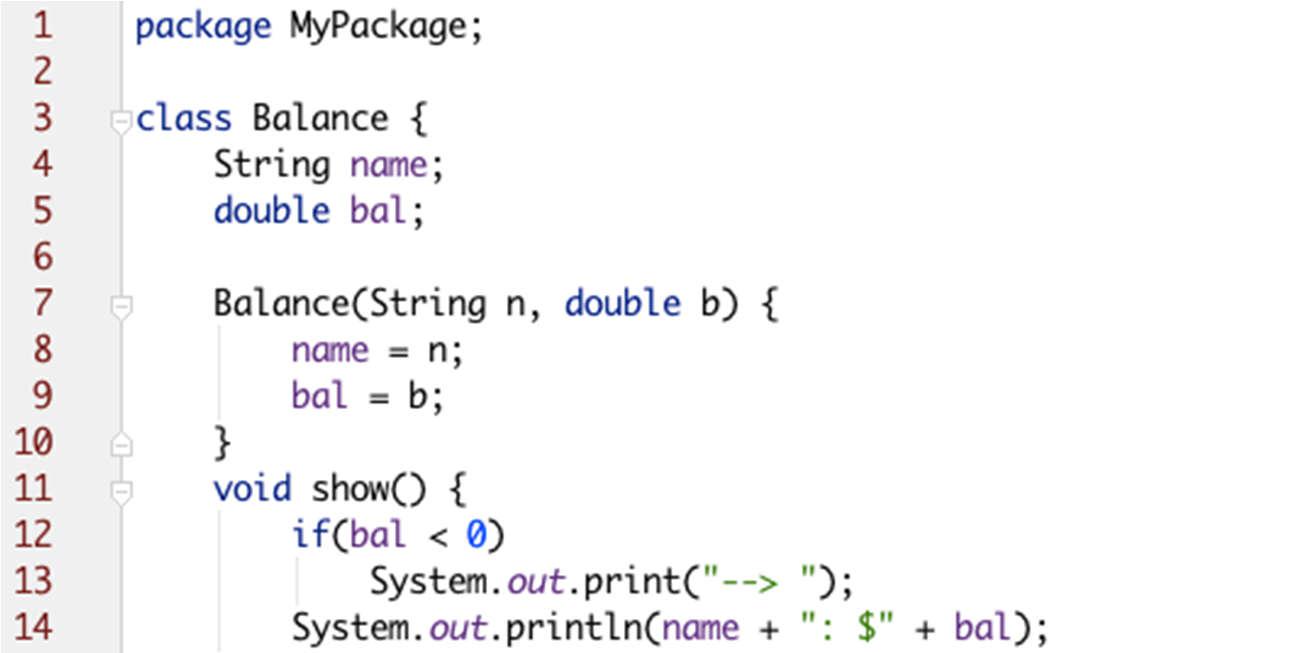
* Java uses file system directories to store packages

– the .class files for any classes that are part of MyPackage must be stored in a directory called MyPackage

* More than one file can include the same package statement
* The package statement simply specifies to which package the classes defined in a file belong
* To create hierarchy of packages, separate each package name from the one above it by use of a (.)

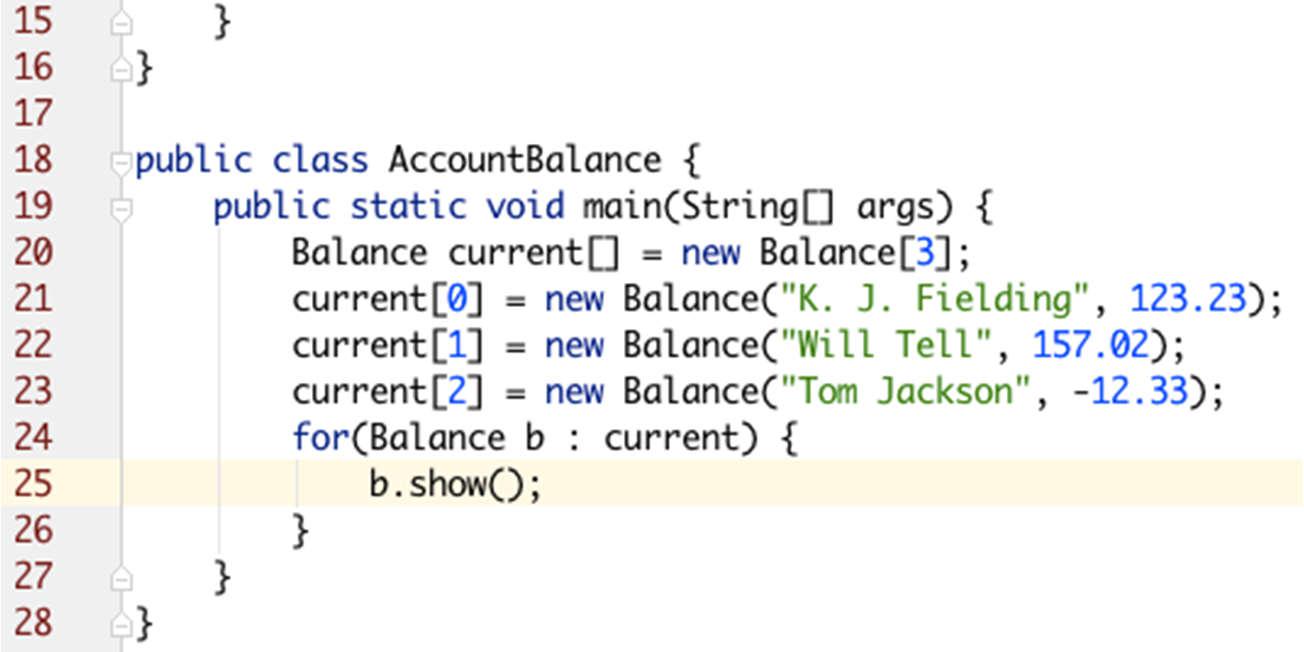
Prepared By - Rifat Shahriyar 5

Package Example



**javac -d . AccountBalance.java**

**java MyPackage.AccountBalance**



Prepared By - Rifat Shahriyar 6

Package Syntax

* The general form of a multilevel package statement

– ***package pkg1[.pkg2[.pkg3]]***

– ***package java.awt.image***

* In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions
* The general form of the import statement

– ***import pkg1 [.pkg2].(classname | \*)***

– ***import java.util.Scanner***

Prepared By - Rifat Shahriyar 7

Access Protection

* Packages act as containers for classes and other subordinate packages
* Classes act as containers for data and code
* The class is Java’s smallest unit of abstraction
* Four categories of visibility for class members

– Subclasses in the same package

– Non-subclasses in the same package

– Subclasses in different package

– Classes that are neither in the same package nor subclasses

Prepared By - Rifat Shahriyar 8

Access Protection

* The three access modifiers provide a variety of ways to produce the many levels of access required

– private, public, and protected

* The following applies only to members of classes



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Private** | **No Modifier** | **Protected** | **Public** |
|  |  |  |  |  |
| Same class | Yes | Yes | Yes | Yes |
|  |  |  |  |  |
| Same package subclass | No | Yes | Yes | Yes |
|  |  |  |  |  |
| Same package non-subclass | No | Yes | Yes | Yes |
|  |  |  |  |  |
| Different package subclass | No | No | Yes | Yes |
|  |  |  |  |  |
| Different package non-subclass | No | No | No | Yes |
|  |  |  |  |  |

Prepared By - Rifat Shahriyar 9

Access Protection

* Anything declared ***public*** can be accessed from anywhere
* Anything declared ***private*** cannot be seen outside of its class
* When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package (***default access***)
* If you want to allow an element to be seen outside your current package, but only to classes that subclass the class directly, then declare that element ***protected***

Prepared By - Rifat Shahriyar 10

Access Protection

* A non-nested class has only two possible access levels

– **default** and **public** (others are **abstract** and **final**)

* When a class is declared as public, it is accessible by any other code
* If a class has default access, then it can only be accessed by other code within its same package
* When a class is public, it must be the only public class declared in the file, and the file must have the same name as the class

Prepared By - Rifat Shahriyar 11

***Interface***

Prepared By - Rifat Shahriyar 12

Interface

* We can call it a pure abstract class having no concrete methods

– All methods declared in an interface are implicitly **public** and **abstract**

– All variables declared in an interface are implicitly **public**, **static** and **final**

* *An interface can’t have instance variables, so can’t maintain state information unlike class*
* A class can only extend from a **single class**, but a class can implement **multiple interfaces**

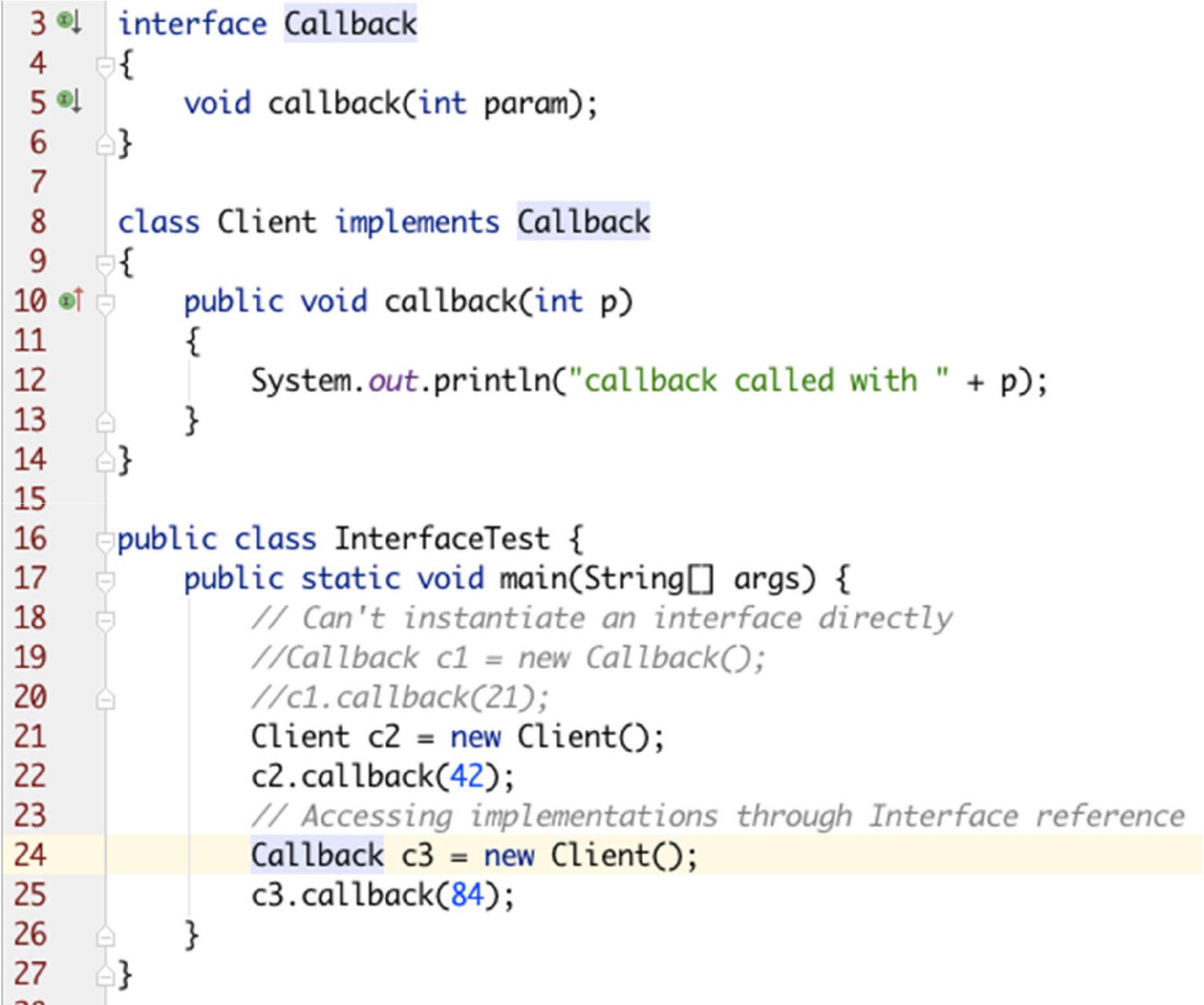
Prepared By - Rifat Shahriyar 13

Implementing Interface

* When you implement an interface method, it must be declared as public
* By implementing an interface, a class signs a contract with the compiler that it will definitely provide implementation of all the methods
* If it fails to do so, the class will be considered as abstract
* Then it must be declared as abstract and no object of that class can be created

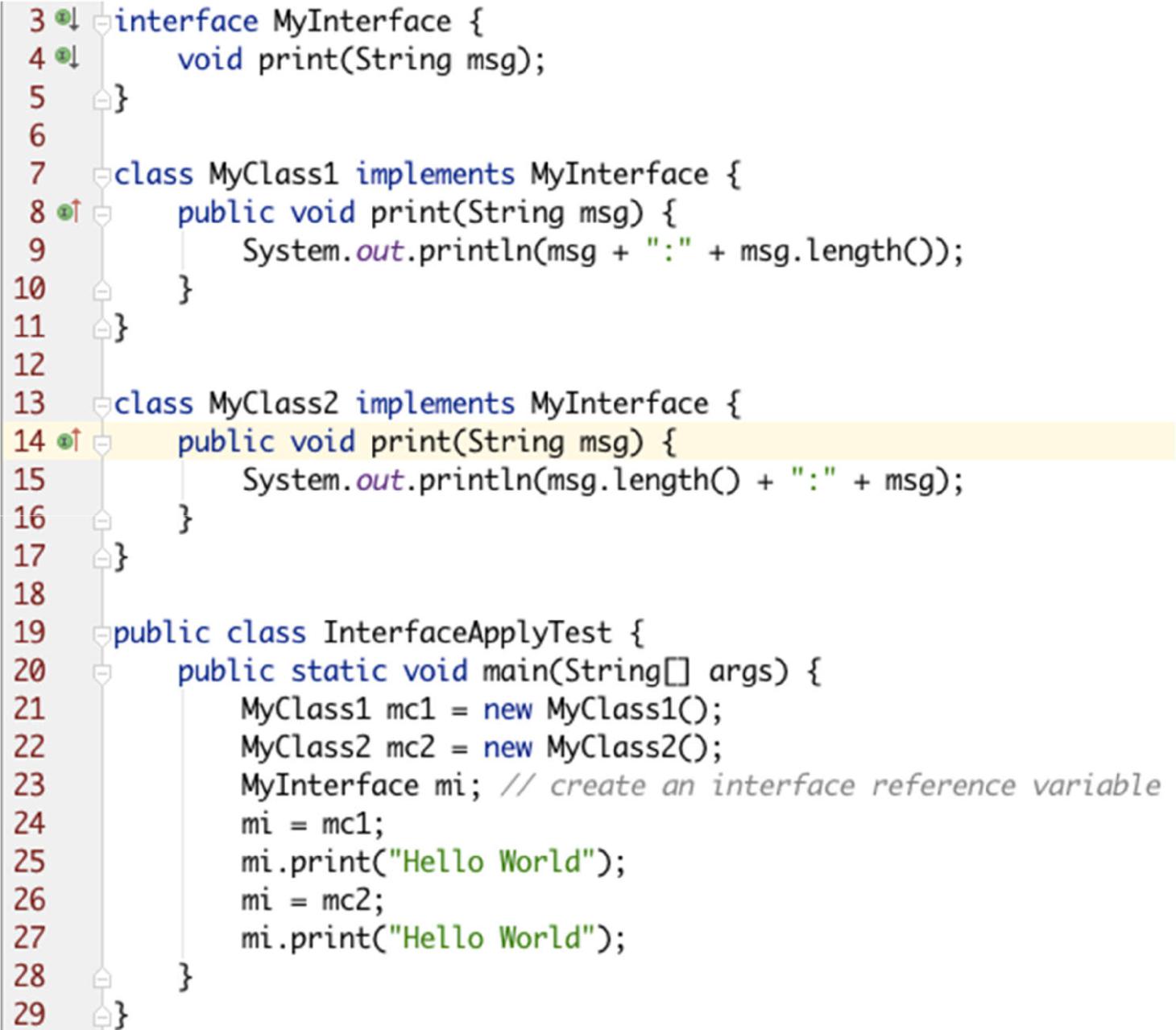
Prepared By - Rifat Shahriyar 14

Simple Interface



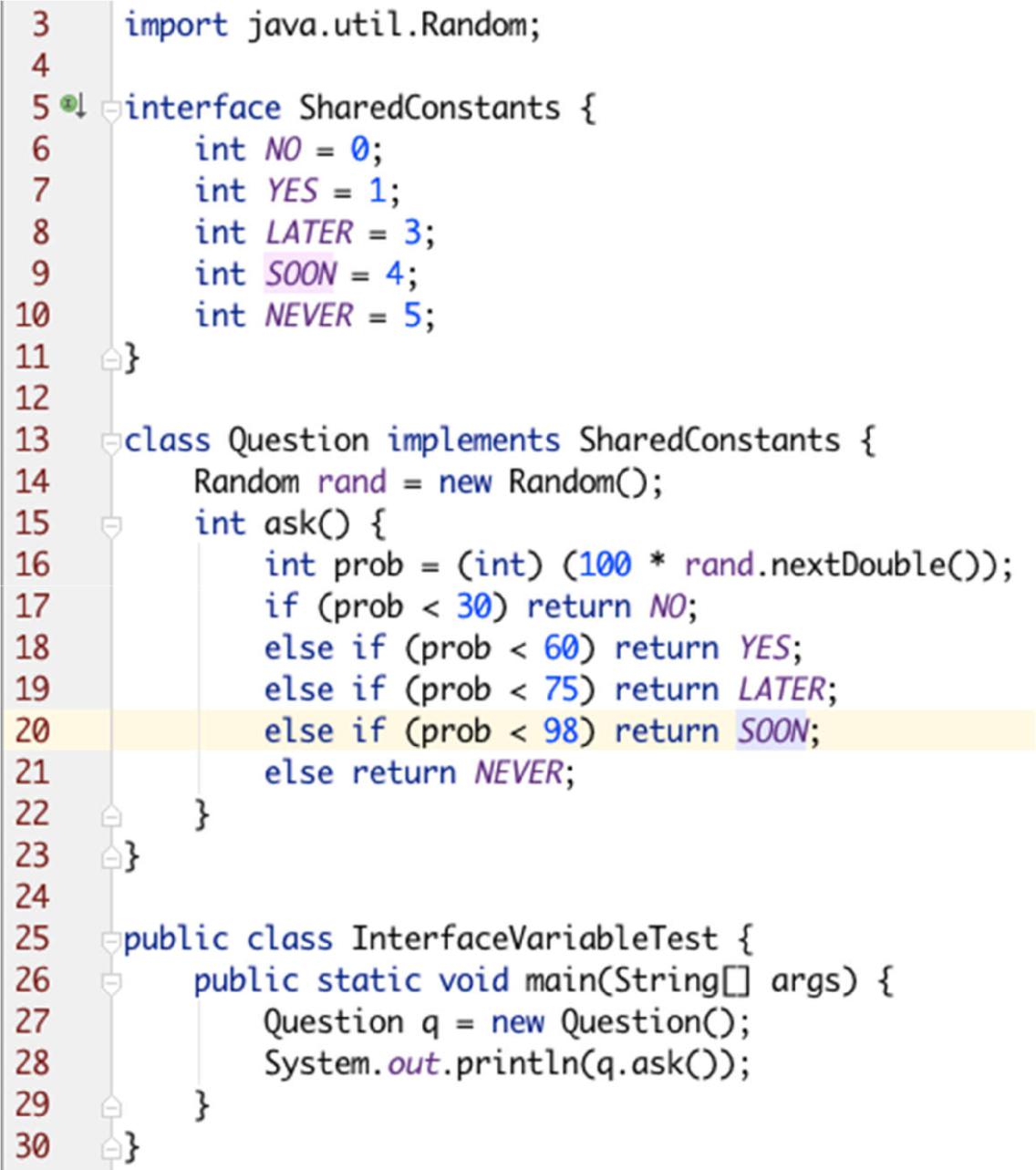
Prepared By - Rifat Shahriyar 15

Applying Interfaces



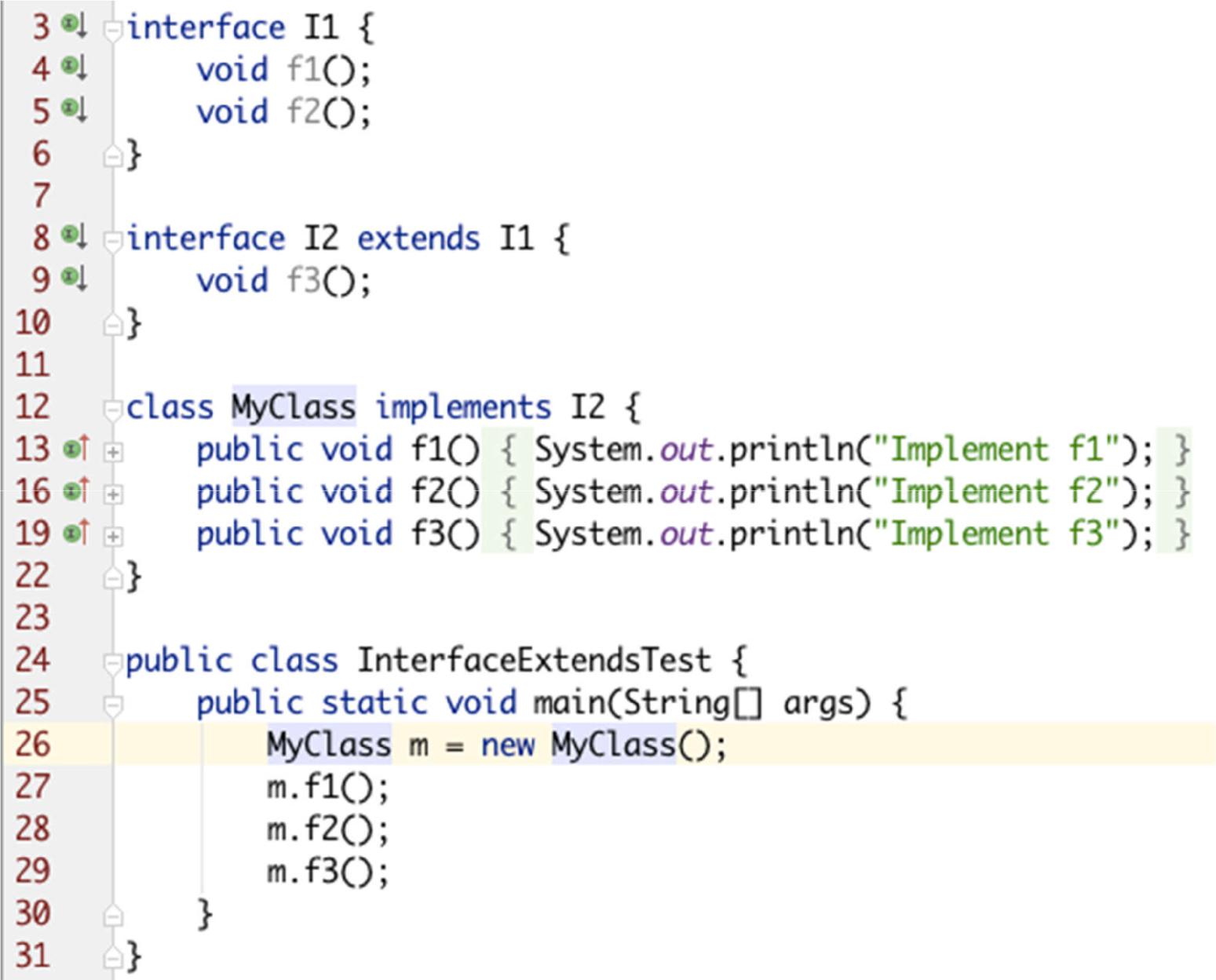
Prepared By - Rifat Shahriyar 16

Variables in Interfaces



Prepared By - Rifat Shahriyar 17

Extending Interfaces



Prepared By - Rifat Shahriyar 18

Default Interface Methods

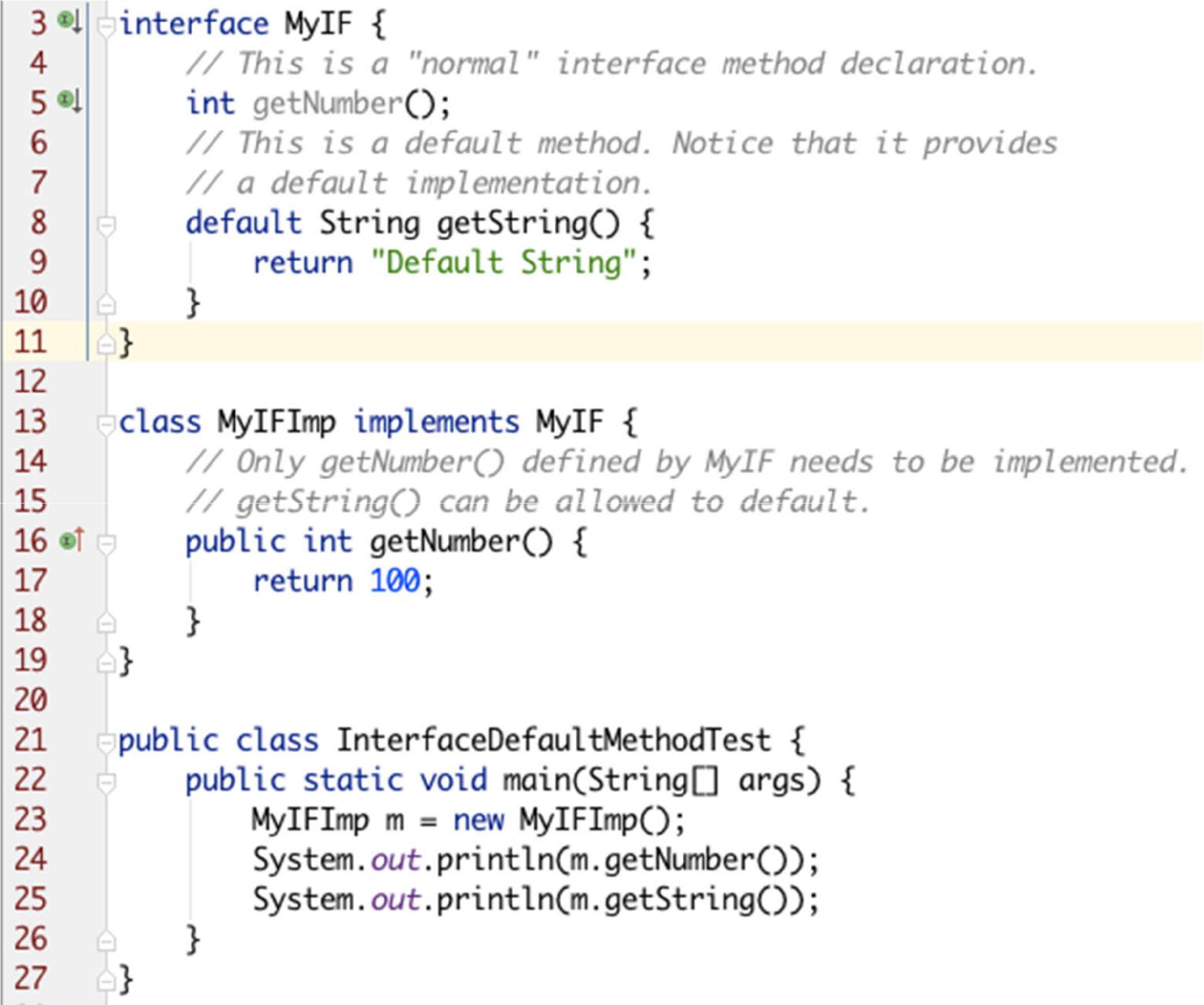
* Prior to JDK 8, an interface could not define any implementation whatsoever
* The release of JDK 8 has changed this by adding a new capability to interface called the *default method*

– A default method lets you define a default implementation for an interface method

– Its primary motivation was to provide a means by which interfaces could be expanded without breaking existing code

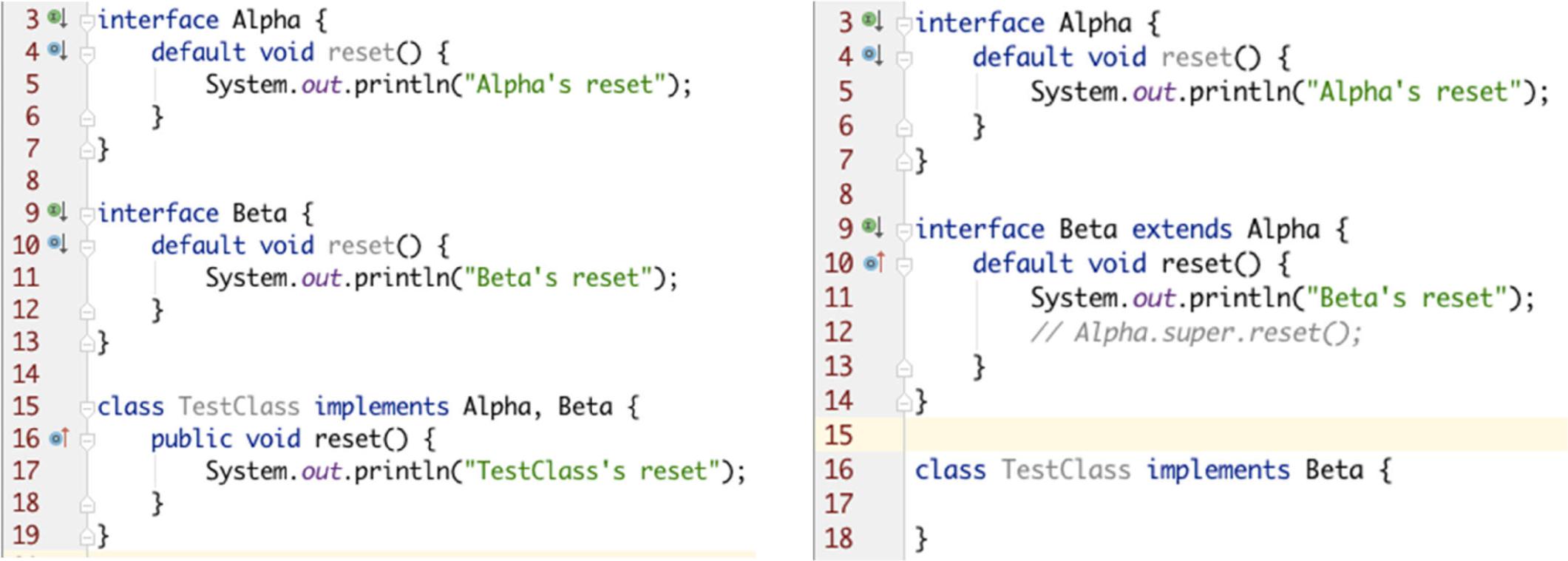
Prepared By - Rifat Shahriyar 19

Default Interface Methods



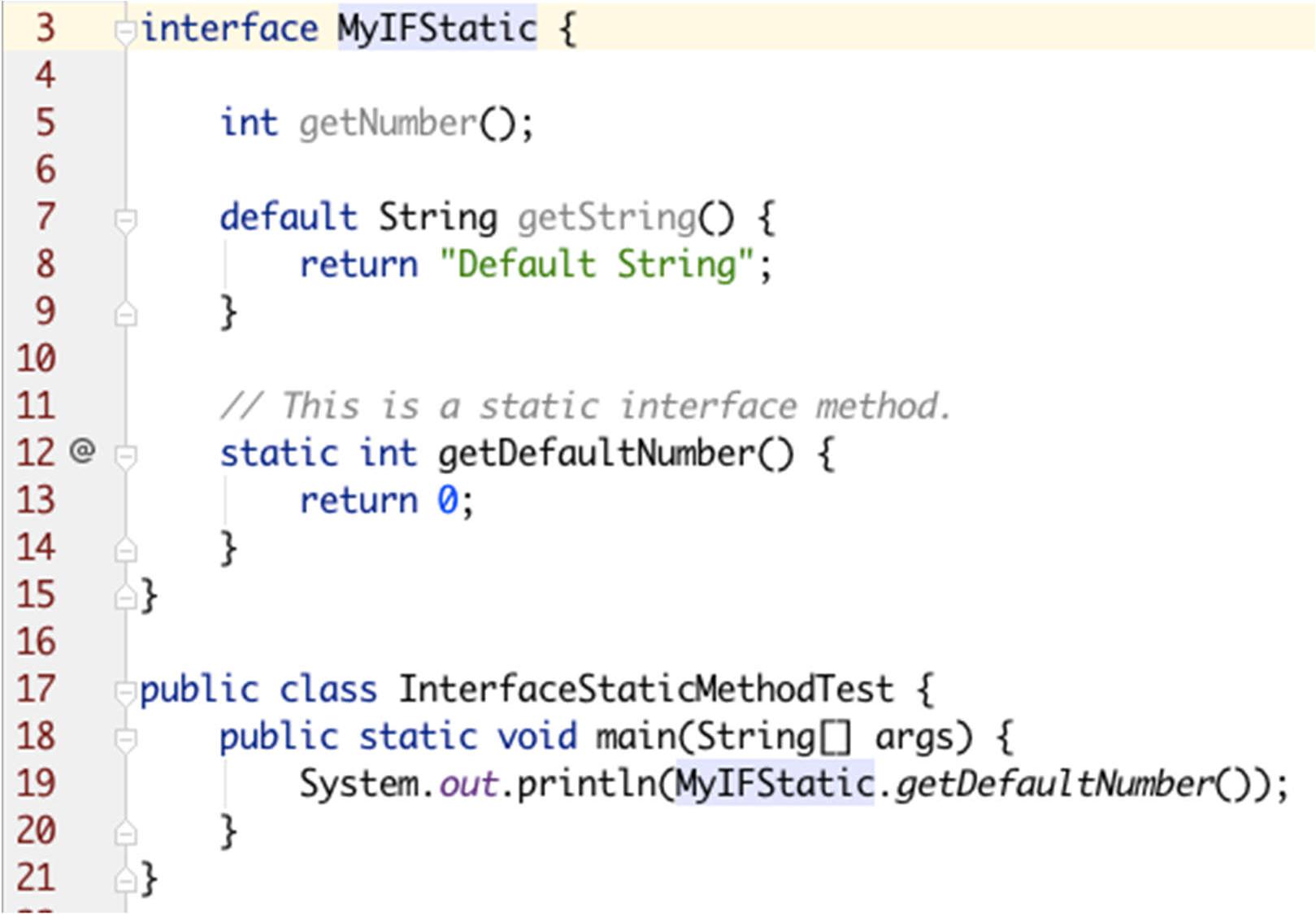
Prepared By - Rifat Shahriyar 20

Multiple Inheritance Issues



Prepared By - Rifat Shahriyar 21

Static Methods in Interface



Prepared By - Rifat Shahriyar 22

***Exception***

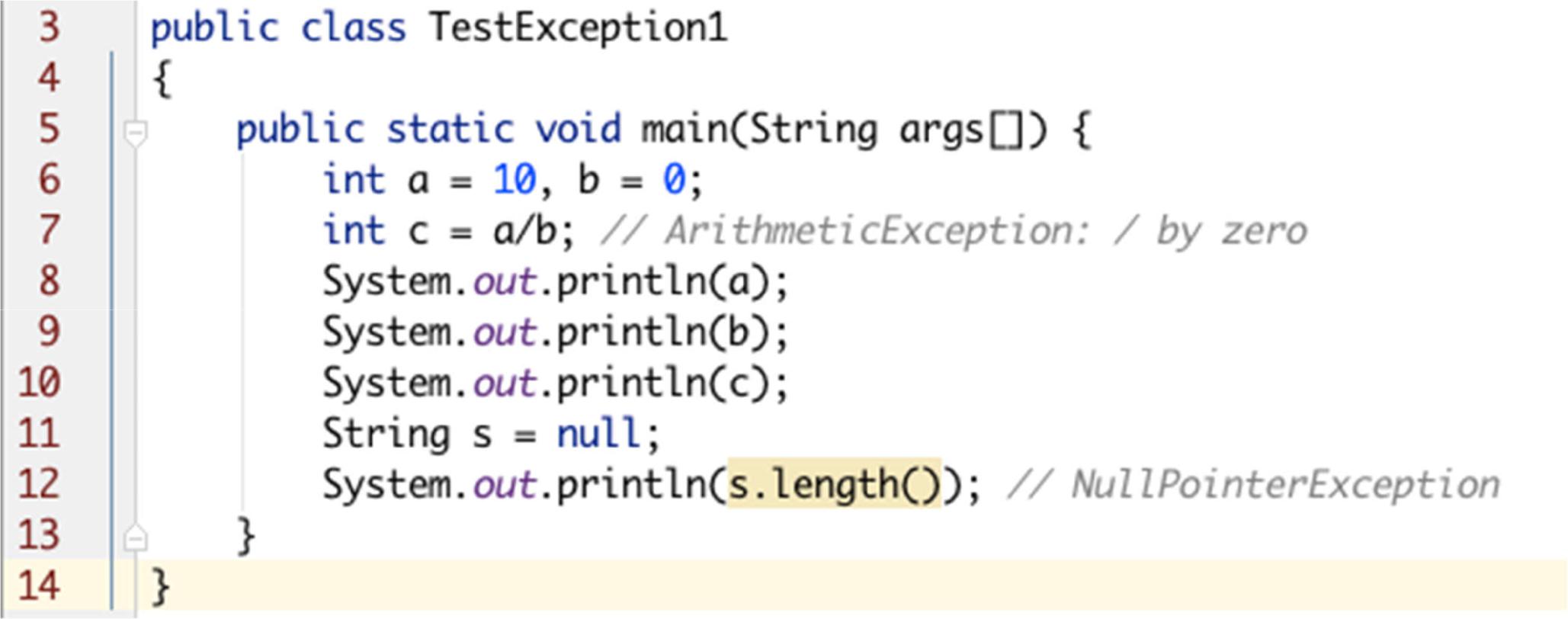
Prepared By - Rifat Shahriyar 23

Exception Handling

* Uncaught exceptions
* Caught exceptions
* try
* catch
* finally
* throw
* throws
* Creating custom exceptions

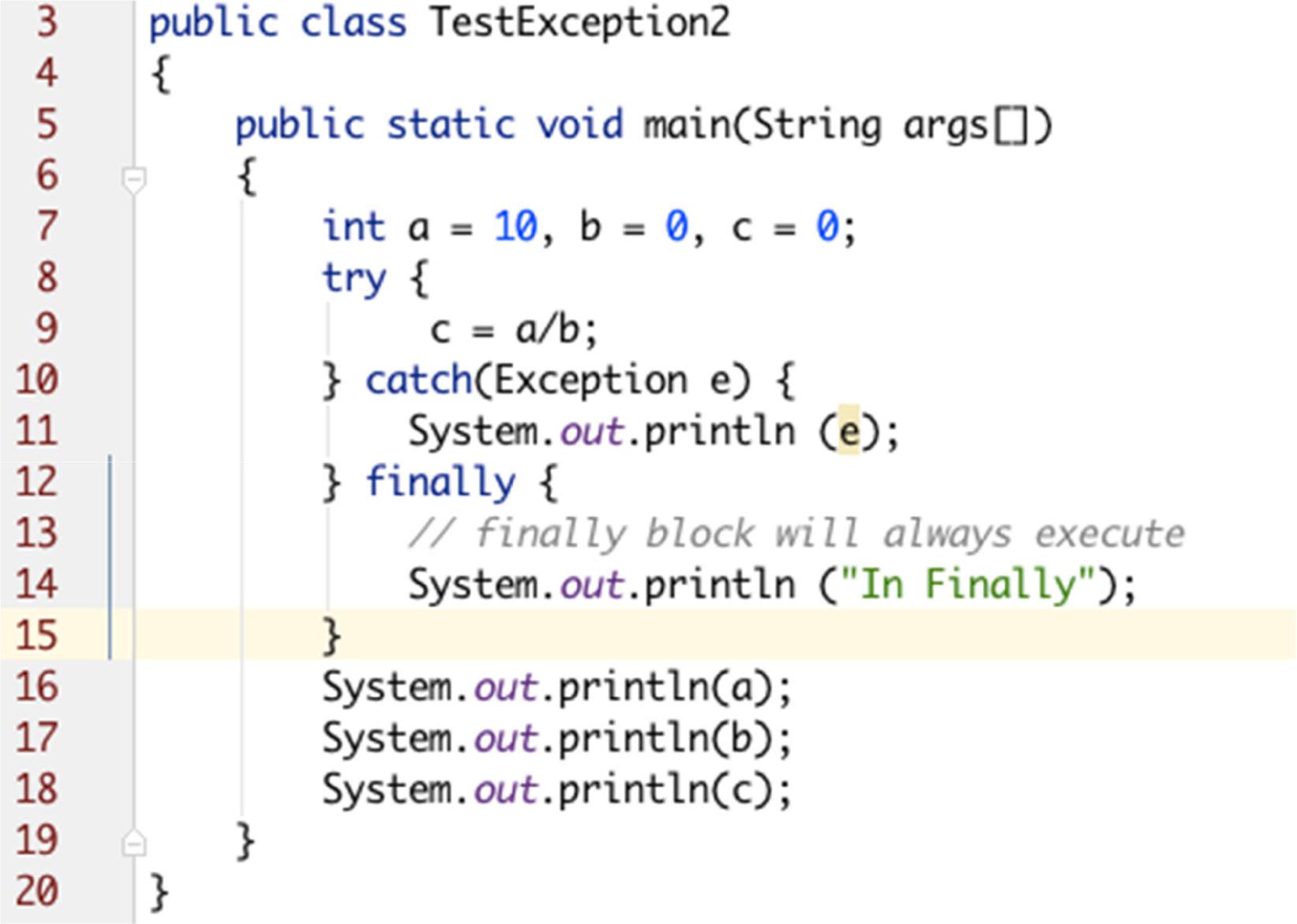
Prepared By - Rifat Shahriyar 24

Uncaught Exceptions



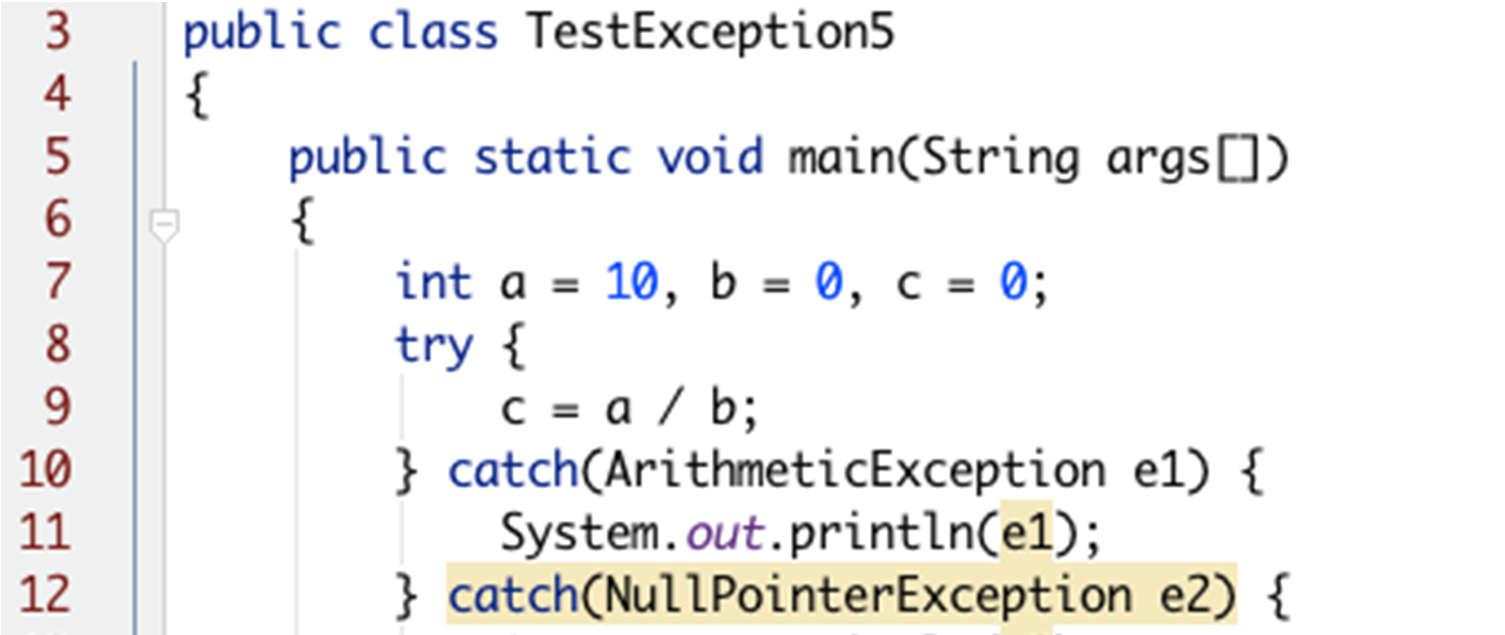
Prepared By - Rifat Shahriyar 25

Caught Exceptions

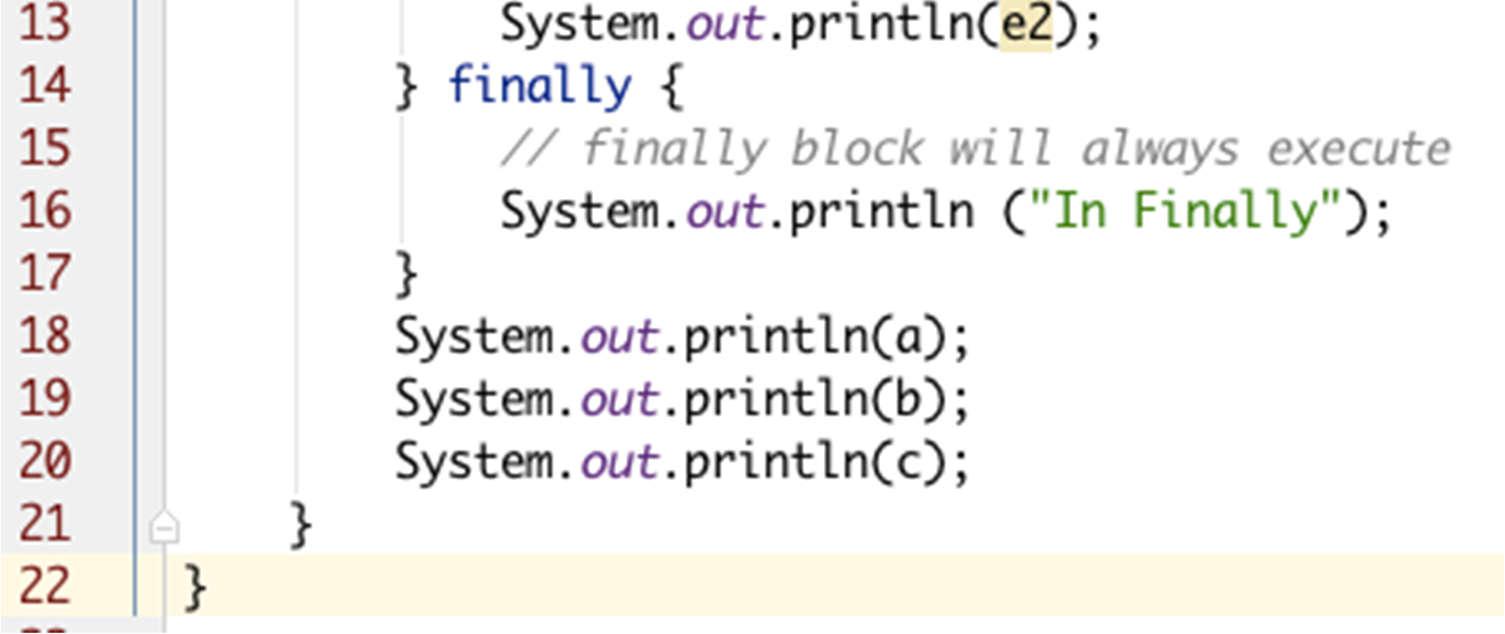


Prepared By - Rifat Shahriyar 26

Caught Exceptions

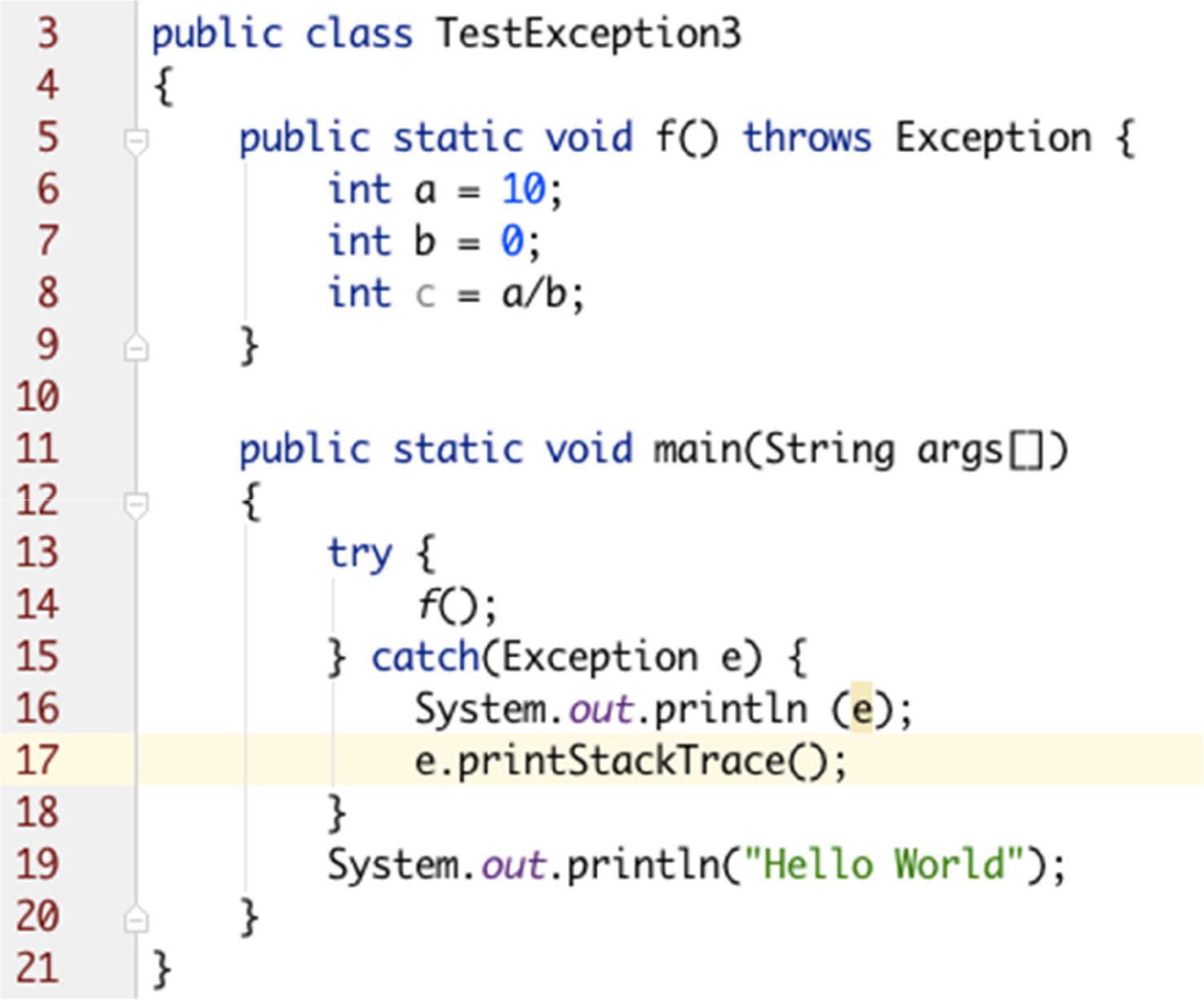


***catch(ArithmeticException | NullPointerException e)***



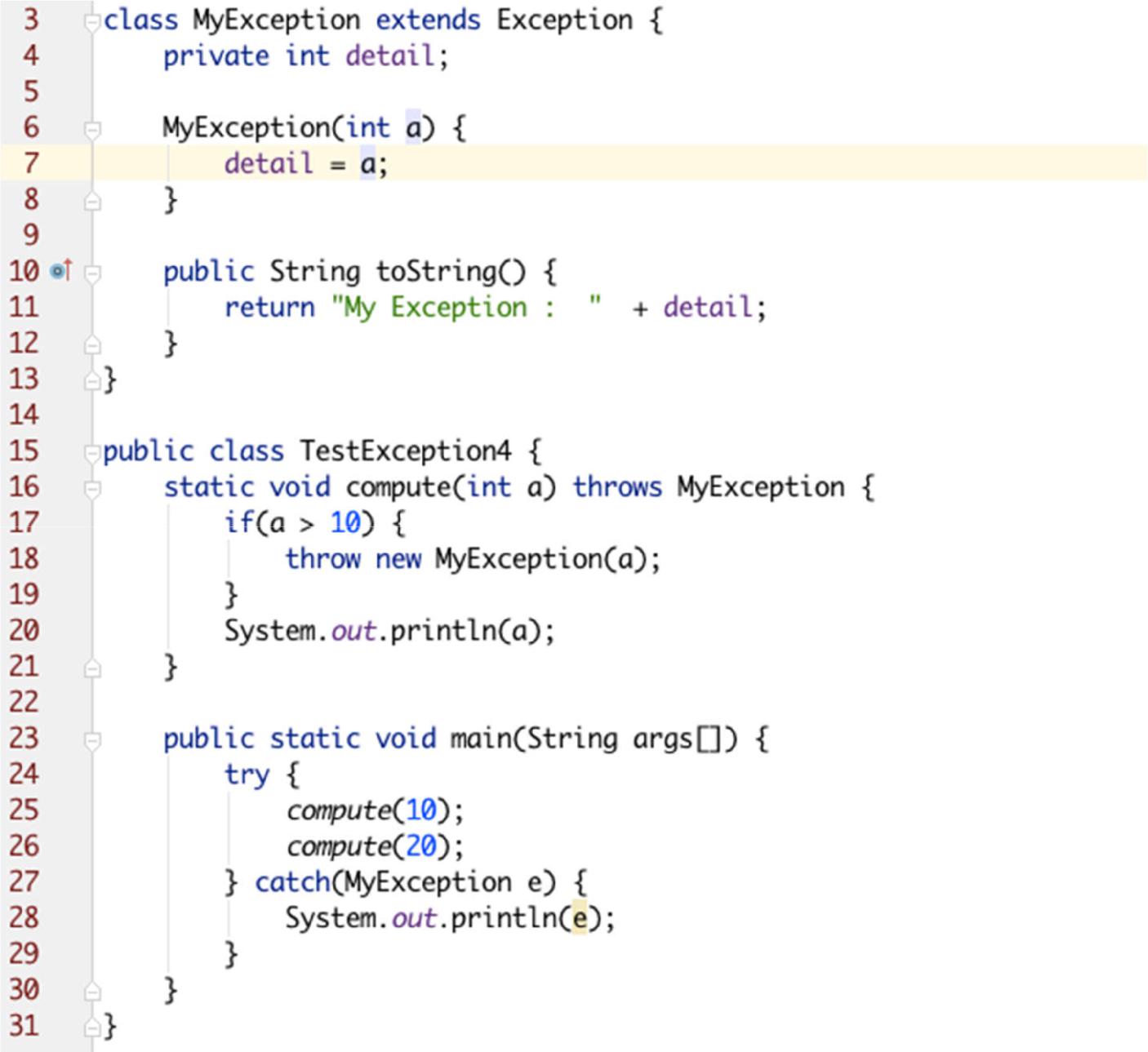
Prepared By - Rifat Shahriyar 27

Throws



Prepared By - Rifat Shahriyar 28

Creating Custom Exceptions



Prepared By - Rifat Shahriyar 29