Java

Package, Interface & Exception

Package

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

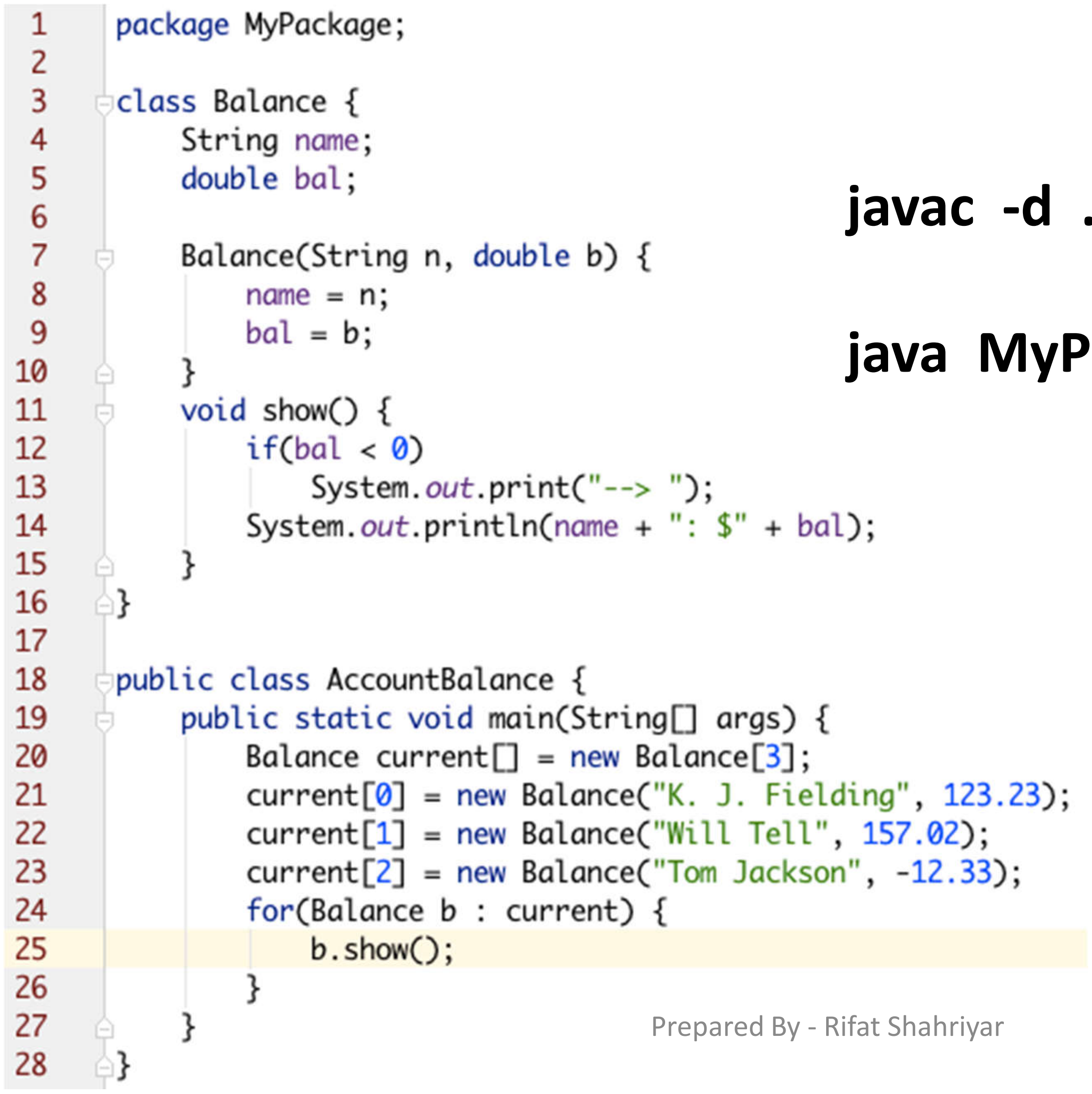
Declaring Package

- ***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

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 (.)

Package Example



```java
package MyPackage;

class Balance {
    String name;
    double bal;

    Balance(String n, double b) {
        name = n;
        bal = b;
    }
    void show() {
        if(bal < 0)
            System.out.print("--> ");
        System.out.println(name + ": $" + bal);
    }
}

public class AccountBalance {
    public static void main(String[] args) {
        Balance current[] = new Balance[3];
        current[0] = new Balance("K. J. Fielding", 123.23);
        current[1] = new Balance("Will Tell", 157.02);
        current[2] = new Balance("Tom Jackson", -12.33);
        for(Balance b : current) {
            b.show();
        }
    }
}
```

javac -d . AccountBalance.java

java MyPackage.AccountBalance

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***

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

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

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***

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

Interface

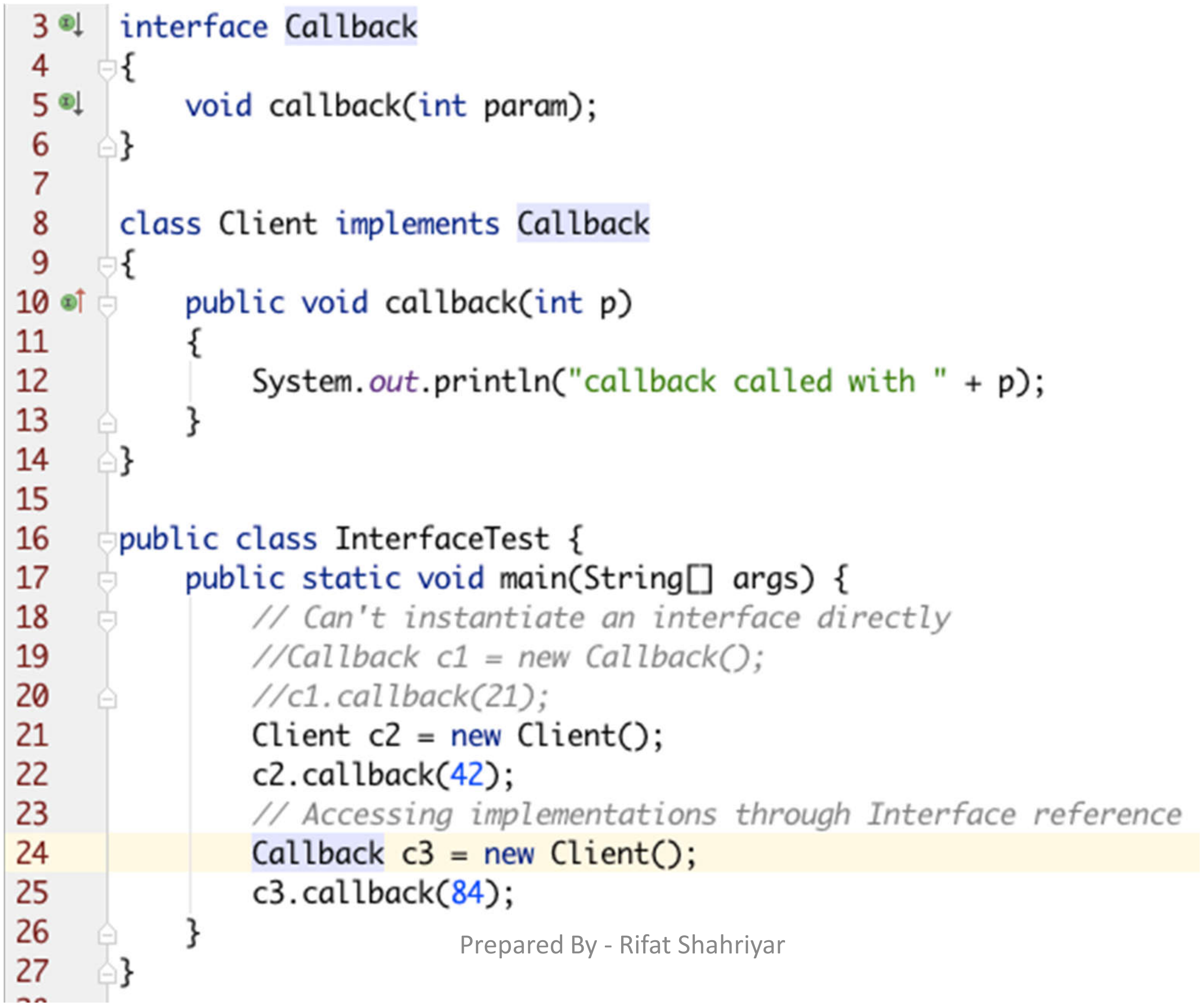
Interface

- 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**

Implementing Interface

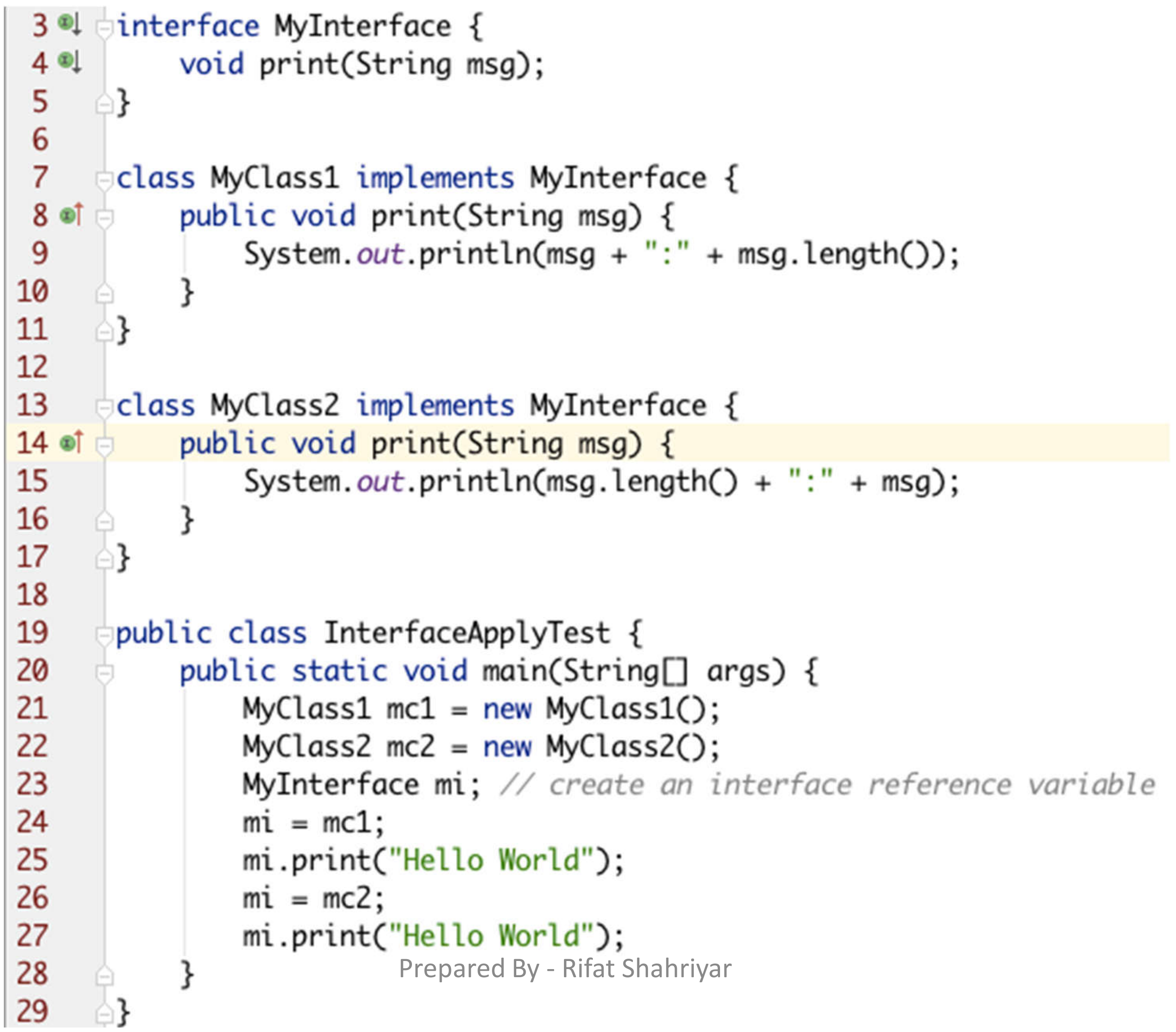
- When you implement an interface method, it must be declared as public
- By implementing an interface, a class signs a contract with the compiler that it will definitely provide implementation of all the methods
- If it fails to do so, the class will be considered as abstract
- Then it must be declared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

Simple Interface



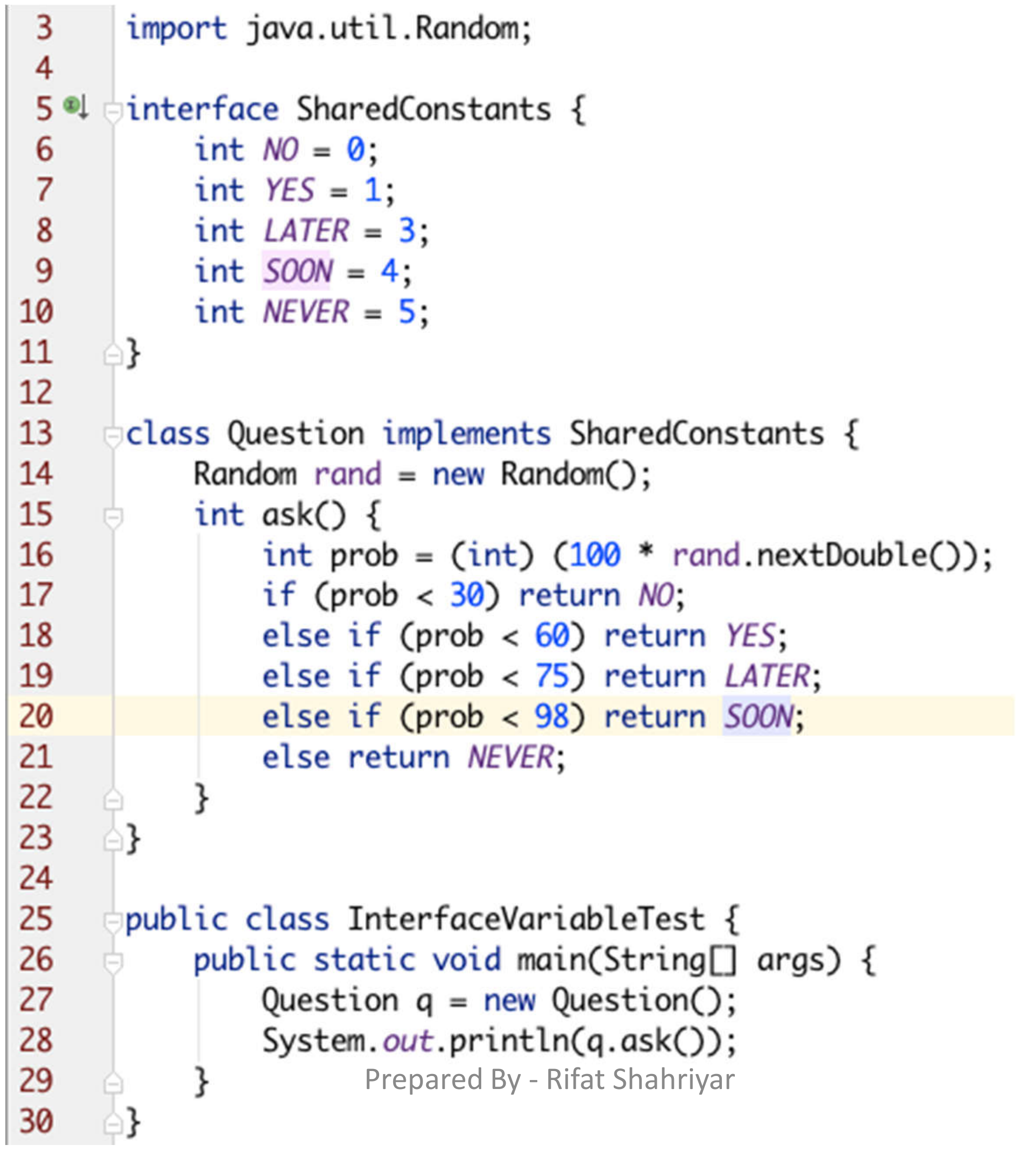
```
interface Callback
{
    void callback(int param);
}

class Client implements Callback
{
    public void callback(int p)
    {
        System.out.println("callback called with " + p);
    }
}

public class InterfaceTest {
    public static void main(String[] args) {
        // Can't instantiate an interface directly
        //Callback c1 = new Callback();
        //c1.callback(21);
        Client c2 = new Client();
        c2.callback(42);
        // Accessing implementations through Interface reference
        Callback c3 = new Client();
        c3.callback(84);
    }
}
```

Applying Interfaces



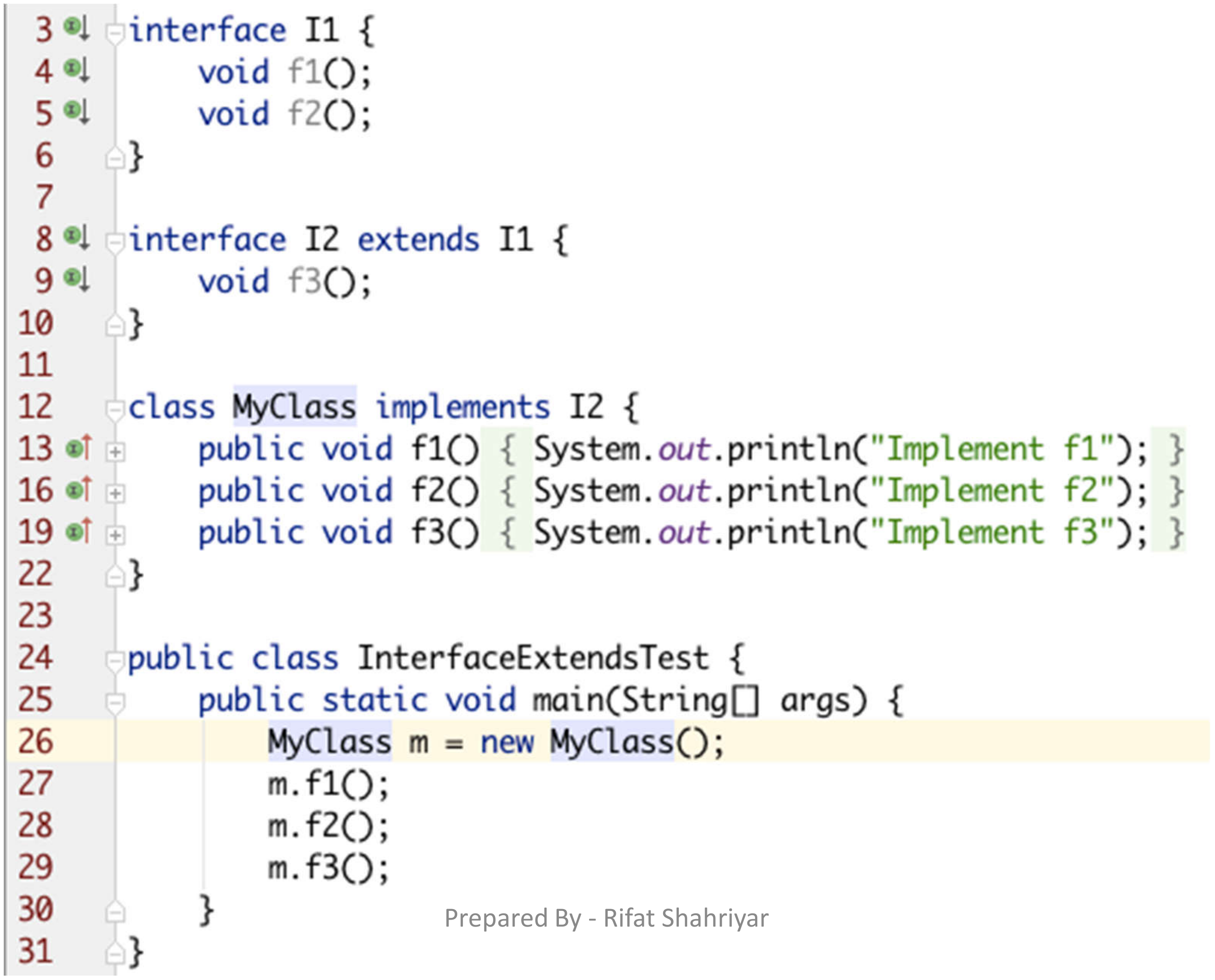
```java
interface MyInterface {
    void print(String msg);
}

class MyClass1 implements MyInterface {
    public void print(String msg) {
        System.out.println(msg + ":" + msg.length());
    }
}

class MyClass2 implements MyInterface {
    public void print(String msg) {
        System.out.println(msg.length() + ":" + msg);
    }
}

public class InterfaceApplyTest {
    public static void main(String[] args) {
        MyClass1 mc1 = new MyClass1();
        MyClass2 mc2 = new MyClass2();
        MyInterface mi; // create an interface reference variable
        mi = mc1;
        mi.print("Hello World");
        mi = mc2;
        mi.print("Hello World");
    }
}
```

Variables in Interfaces



```java
import java.util.Random;

interface SharedConstants {
    int NO = 0;
    int YES = 1;
    int LATER = 3;
    int SOON = 4;
    int NEVER = 5;
}

class Question implements SharedConstants {
    Random rand = new Random();
    int ask() {
        int prob = (int) (100 * rand.nextDouble());
        if (prob < 30) return NO;
        else if (prob < 60) return YES;
        else if (prob < 75) return LATER;
        else if (prob < 98) return SOON;
        else return NEVER;
    }
}

public class InterfaceVariableTest {
    public static void main(String[] args) {
        Question q = new Question();
        System.out.println(q.ask());
    }
}
```

Extending Interfaces

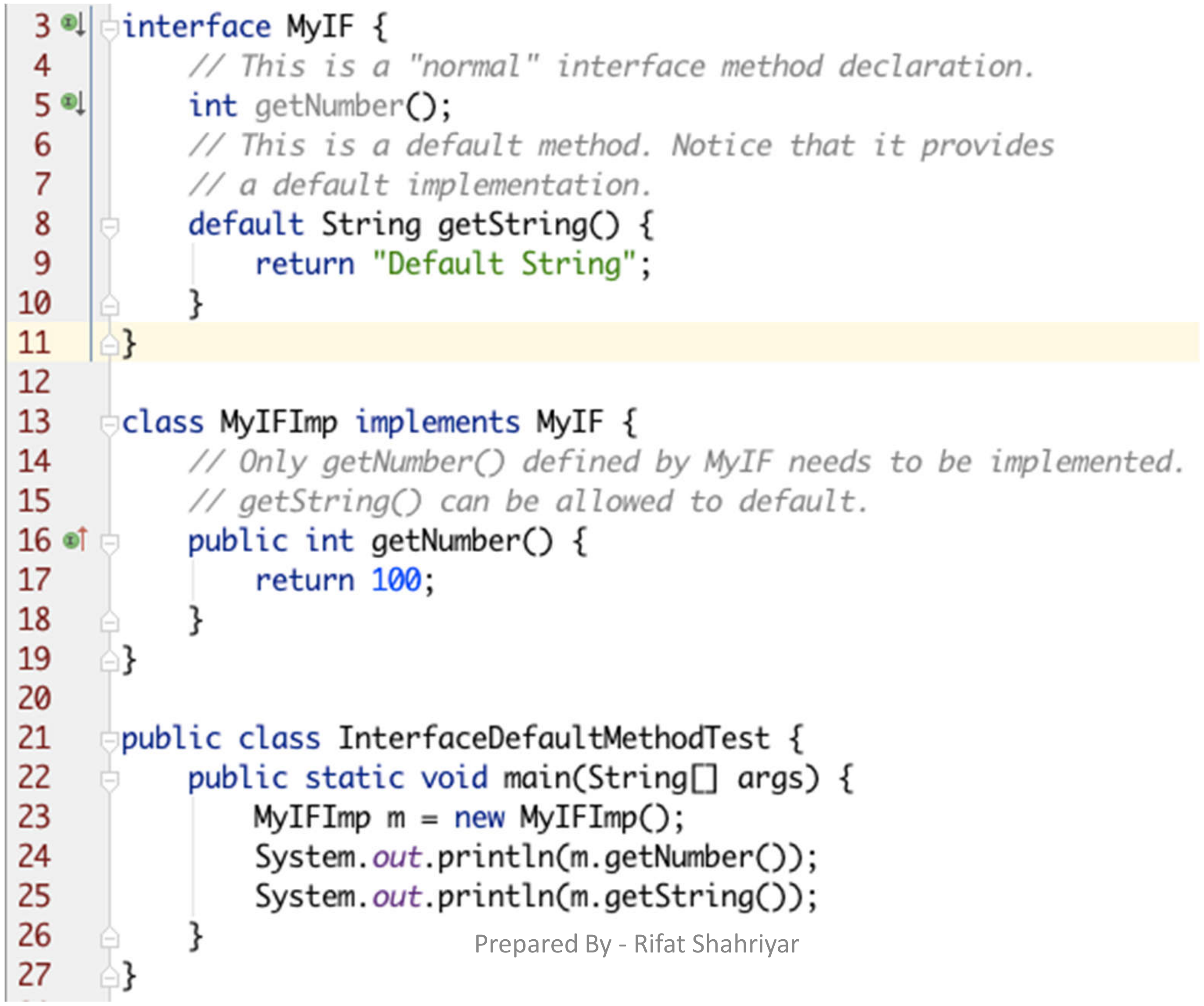


```
interface I1 {
    void f1();
    void f2();
}

interface I2 extends I1 {
    void f3();
}

class MyClass implements I2 {
    public void f1() { System.out.println("Implement f1"); }
    public void f2() { System.out.println("Implement f2"); }
    public void f3() { System.out.println("Implement f3"); }
}

public class InterfaceExtendsTest {
    public static void main(String[] args) {
        MyClass m = new MyClass();
        m.f1();
        m.f2();
        m.f3();
    }
}
```

Default Interface Methods

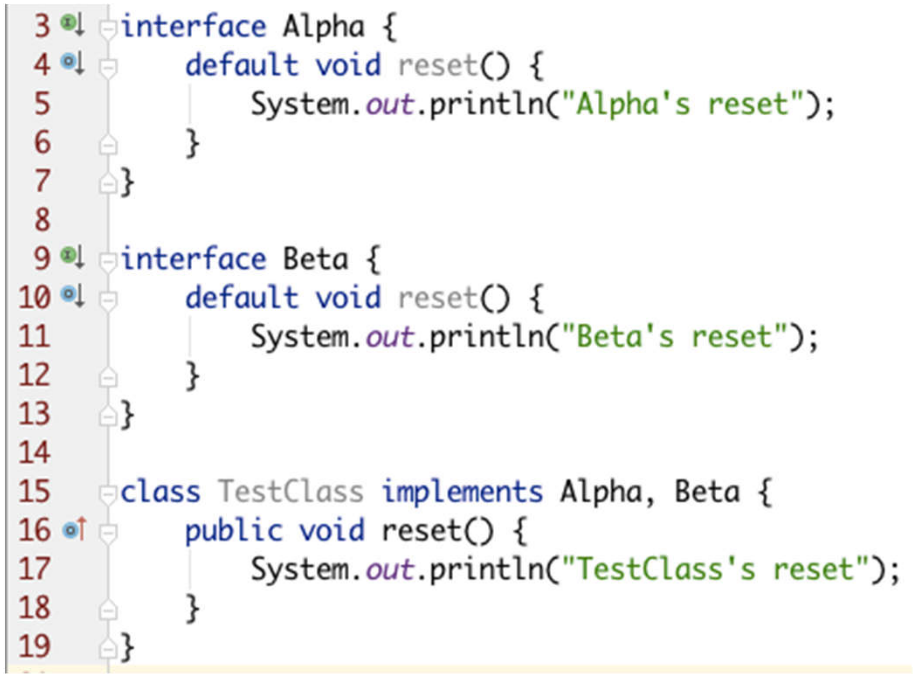
- Prior to JDK 8, an interface could not define any implementation whatsoever
- The release of JDK 8 has changed this by adding a new capability to interface called the *default method*
  - A default method lets you define a default implementation for an interface method
  - Its primary motivation was to provide a means by which interfaces could be expanded without breaking existing code

Default Interface Methods

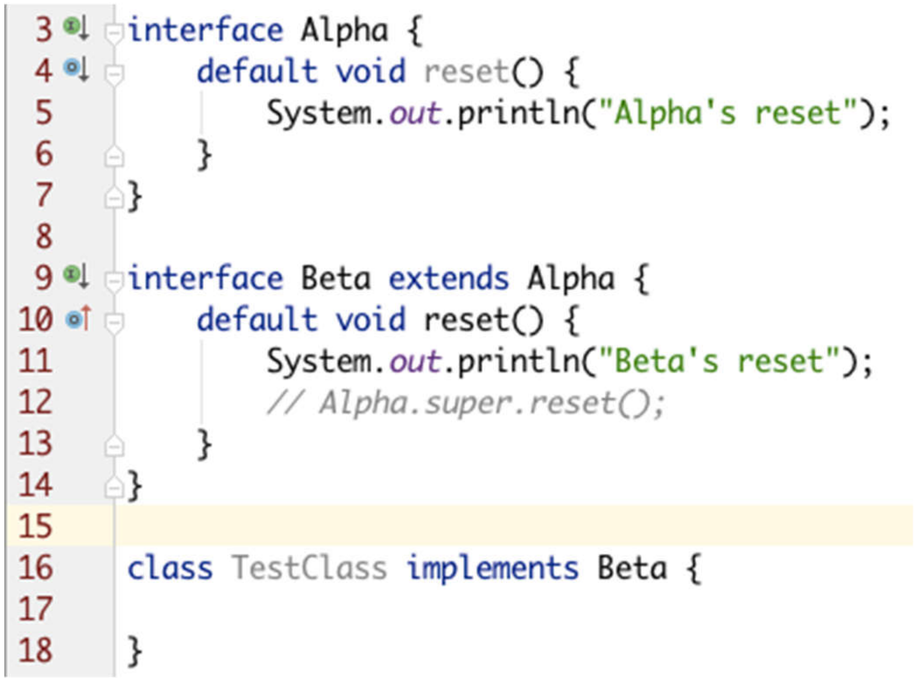


```java
interface MyIF {
    // This is a "normal" interface method declaration.
    int getNumber();
    // This is a default method. Notice that it provides
    // a default implementation.
    default String getString() {
        return "Default String";
    }
}

class MyIFImp implements MyIF {
    // Only getNumber() defined by MyIF needs to be implemented.
    // getString() can be allowed to default.
    public int getNumber() {
        return 100;
    }
}

public class InterfaceDefaultMethodTest {
    public static void main(String[] args) {
        MyIFImp m = new MyIFImp();
        System.out.println(m.getNumber());
        System.out.println(m.getString());
    }
}
```

Multiple Inheritance Issues

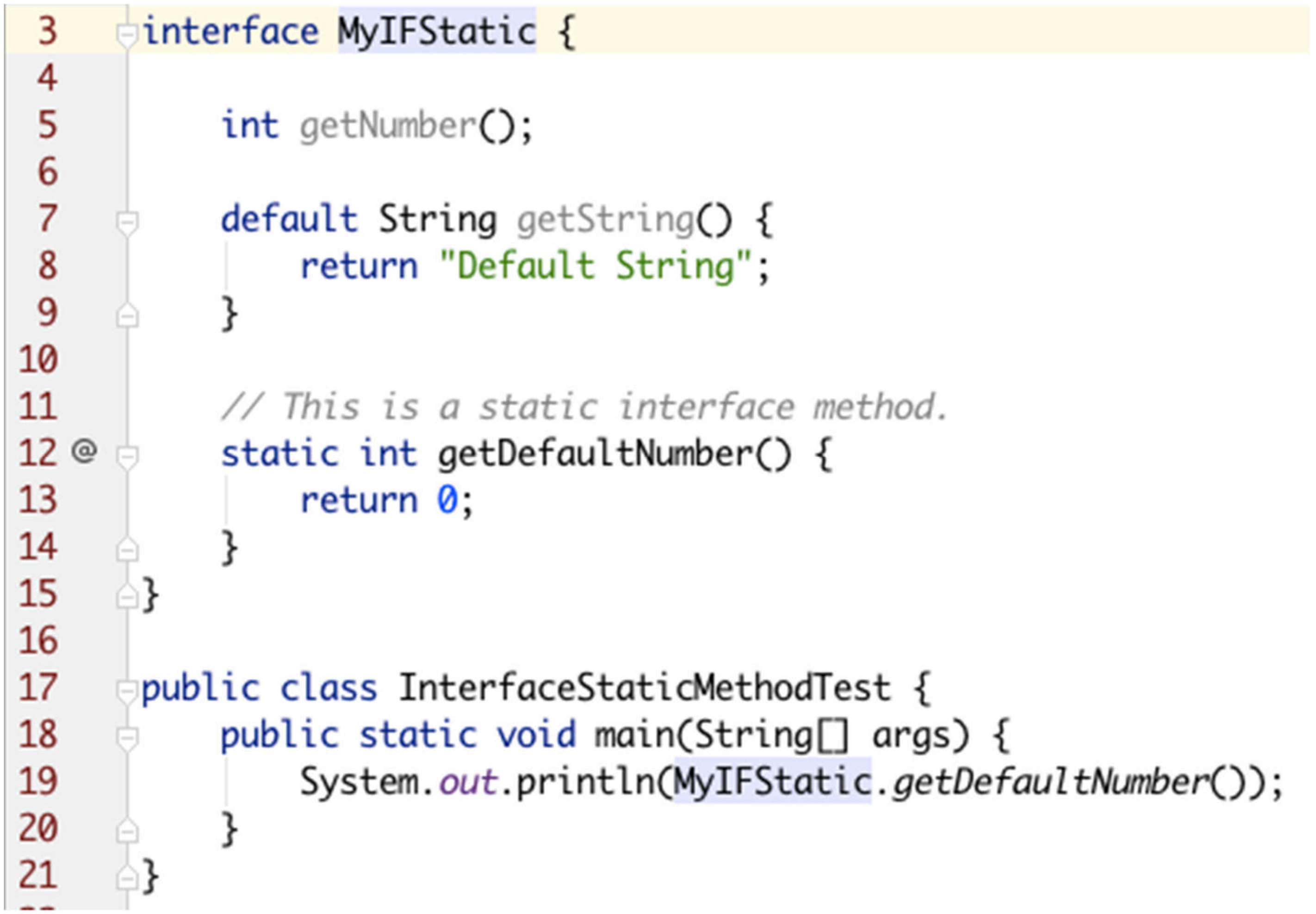


```java
interface Alpha {
    default void reset() {
        System.out.println("Alpha's reset");
    }
}

interface Beta {
    default void reset() {
        System.out.println("Beta's reset");
    }
}

class TestClass implements Alpha, Beta {
    public void reset() {
        System.out.println("TestClass's reset");
    }
}
```



```java
interface Alpha {
    default void reset() {
        System.out.println("Alpha's reset");
    }
}

interface Beta extends Alpha {
    default void reset() {
        System.out.println("Beta's reset");
        // Alpha.super.reset();
    }
}

class TestClass implements Beta {

}
```

Static Methods in Interface



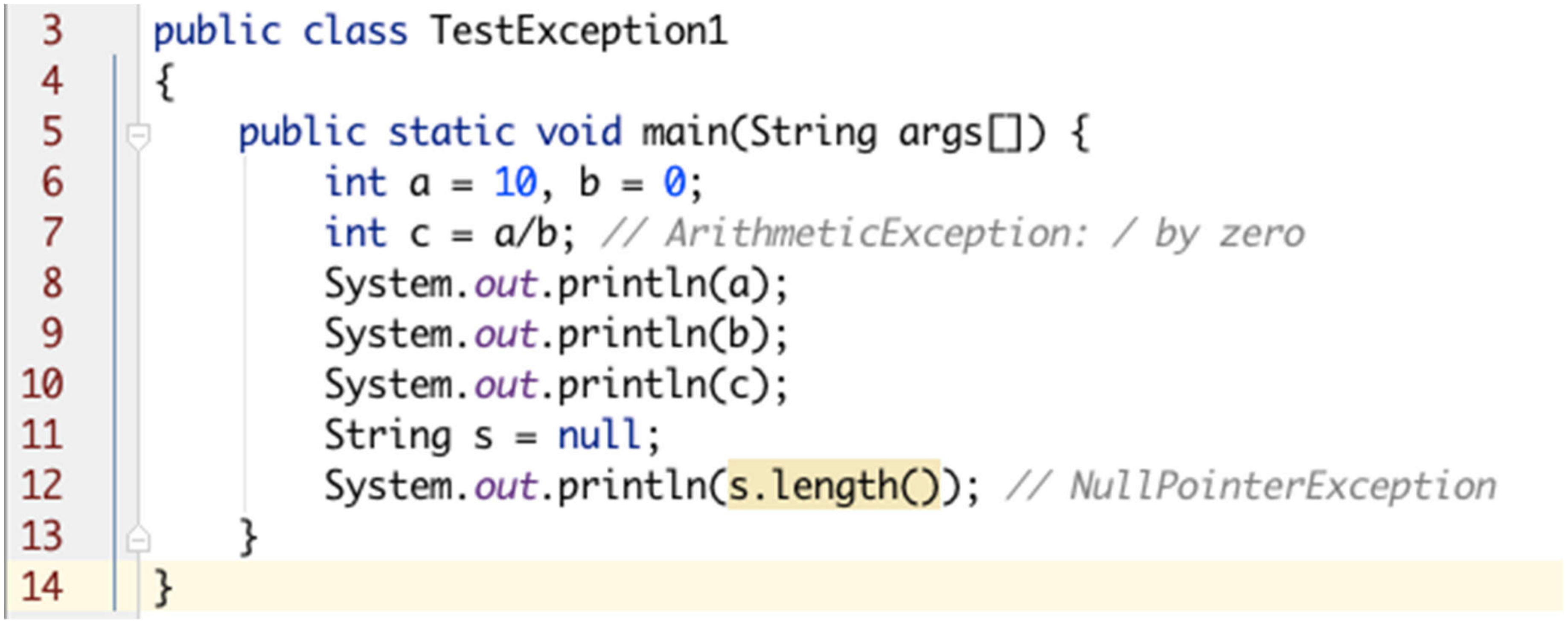
```java
interface MyIFStatic {

    int getNumber();

    default String getString() {
        return "Default String";
    }

    // This is a static interface method.
    static int getDefaultNumber() {
        return 0;
    }
}

public class InterfaceStaticMethodTest {
    public static void main(String[] args) {
        System.out.println(MyIFStatic.getDefaultNumber());
    }
}
```

Exception

Exception Handling

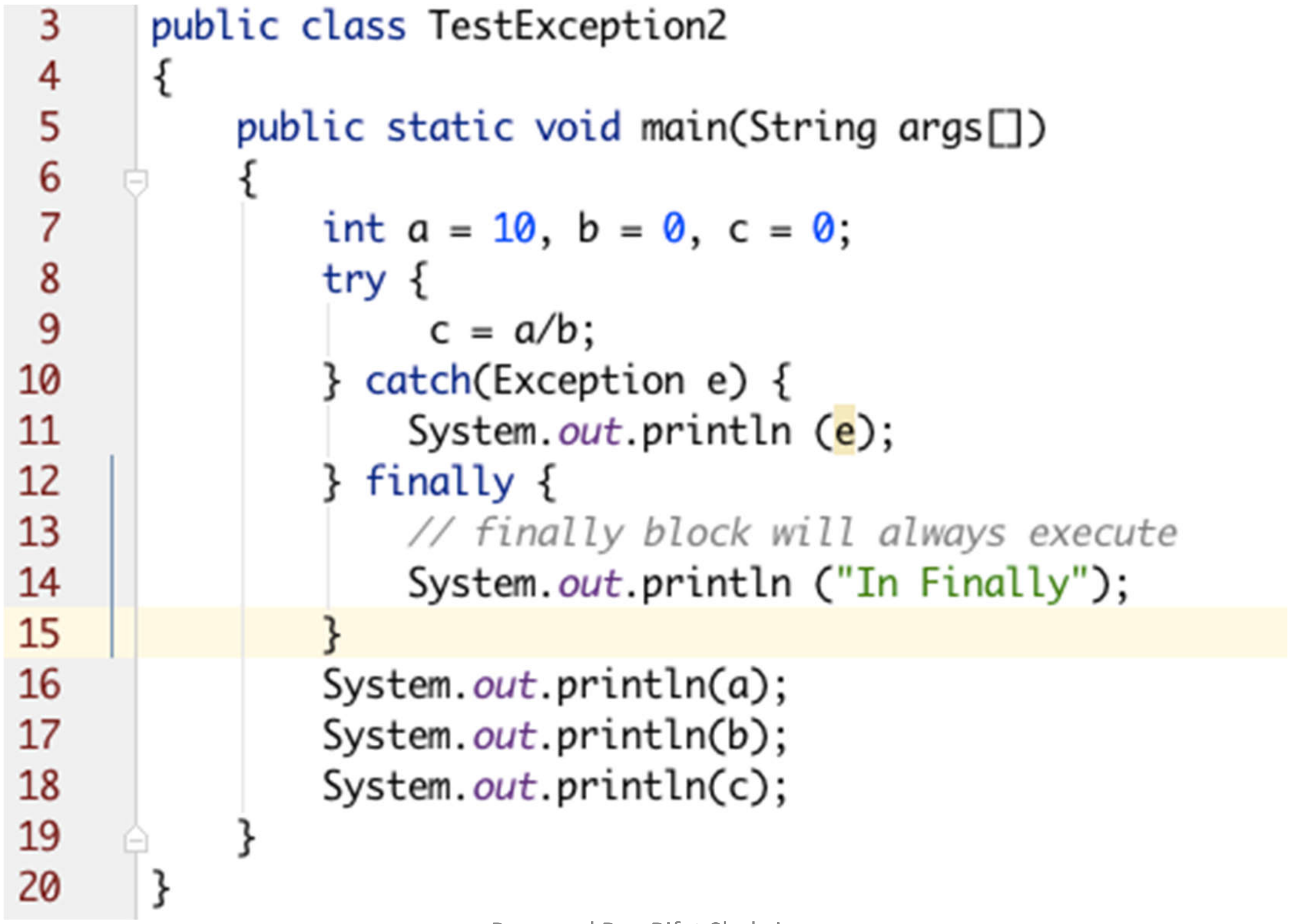
- Uncaught exceptions
- Caught exceptions
- try
- catch
- finally
- throw
- throws
- Creating custom exceptions

Uncaught Exceptions



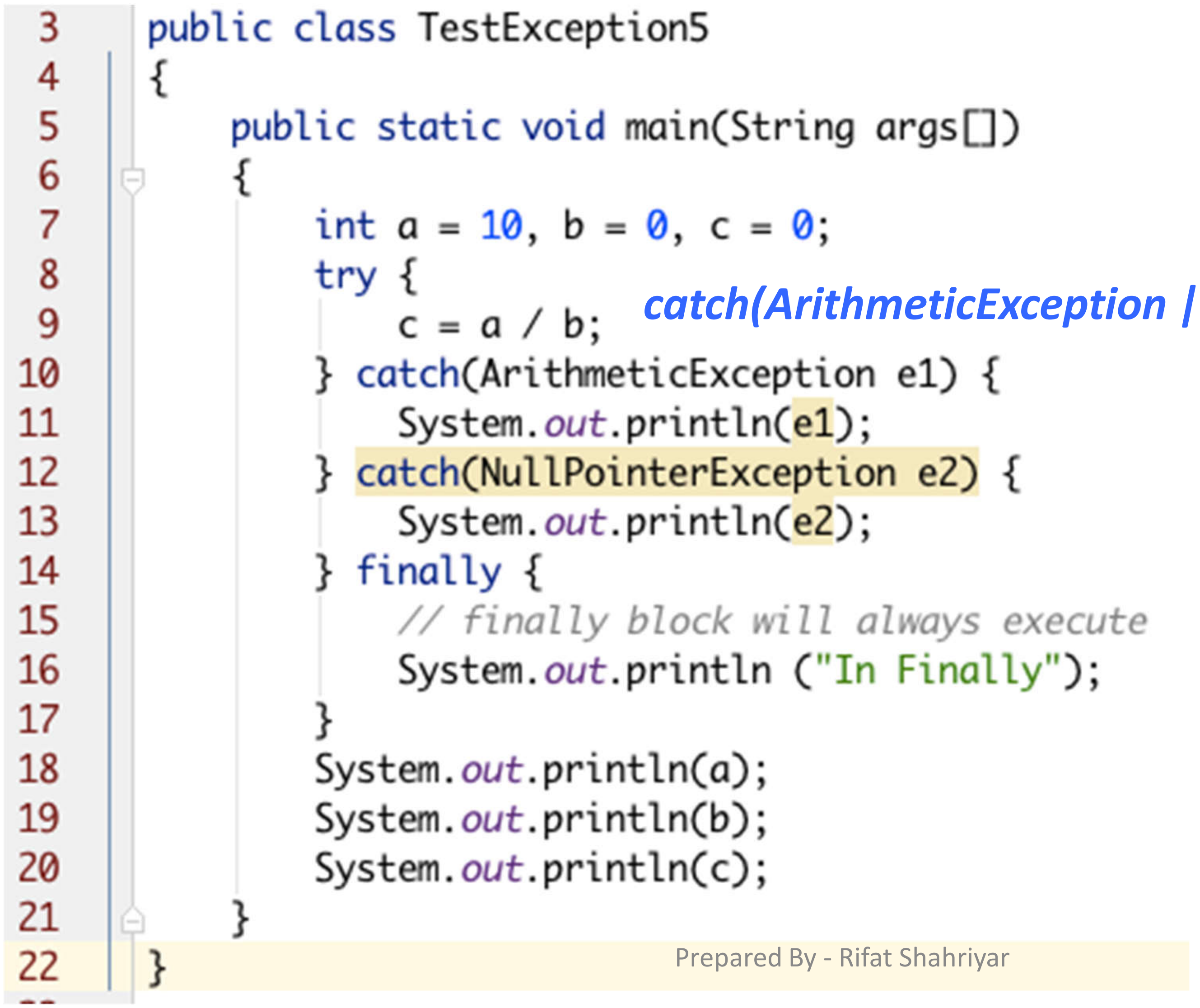
```java
public class TestException1
{
    public static void main(String args[]) {
        int a = 10, b = 0;
        int c = a/b; // ArithmeticException: / by zero
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        String s = null;
        System.out.println(s.length()); // NullPointerException
    }
}
```

Caught Exceptions



```java
public class TestException2
{
    public static void main(String args[])
    {
        int a = 10, b = 0, c = 0;
        try {
             c = a/b;
        } catch(Exception e) {
            System.out.println (e);
        } finally {
            // finally block will always execute
            System.out.println ("In Finally");
        }
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

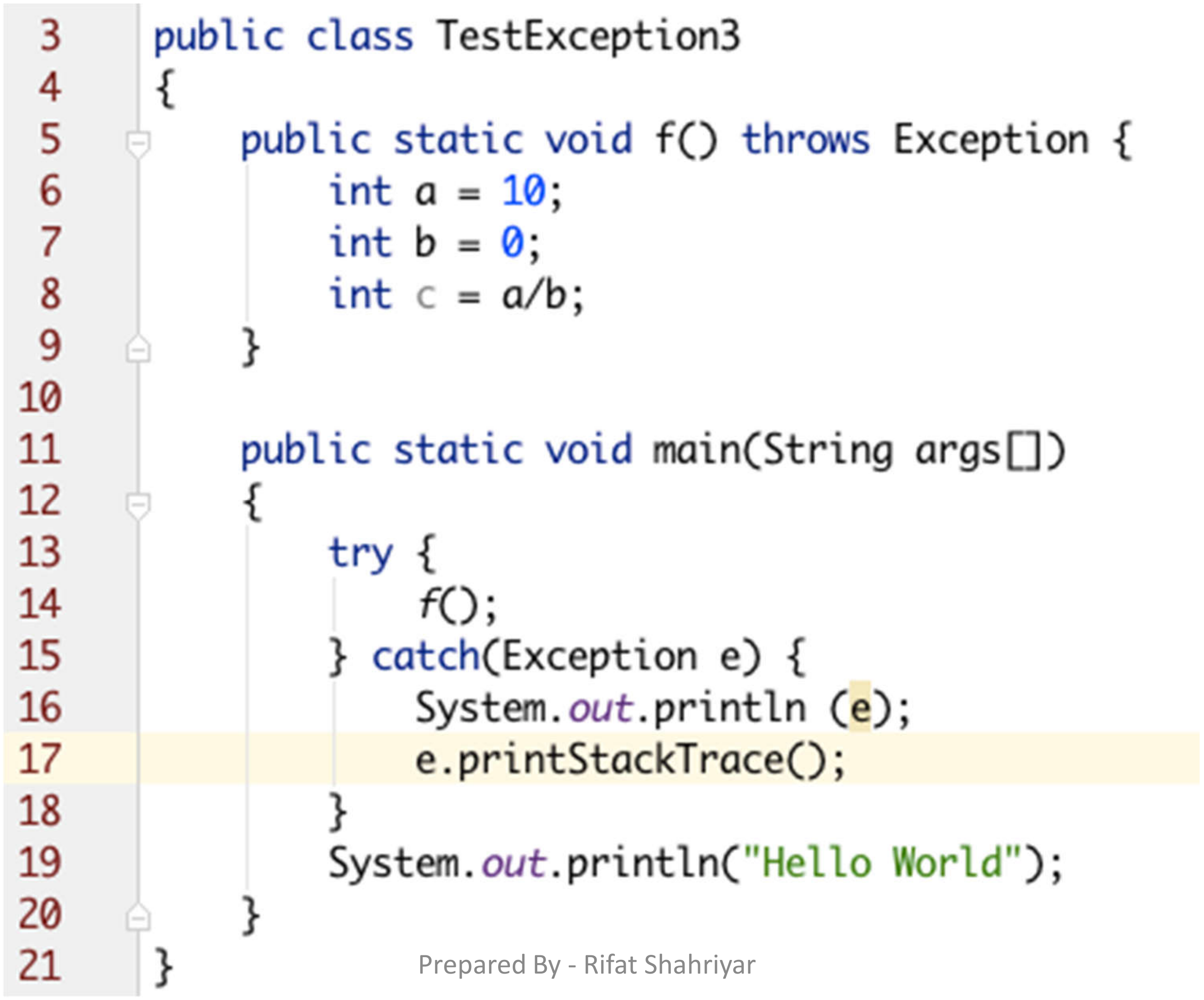
Caught Exceptions



```
public class TestException5
{
    public static void main(String args[])
    {
        int a = 10, b = 0, c = 0;
        try {
            c = a / b;
        } catch(ArithmeticException e1) {
            System.out.println(e1);
        } catch(NullPointerException e2) {
            System.out.println(e2);
        } finally {
            // finally block will always execute
            System.out.println ("In Finally");
        }
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

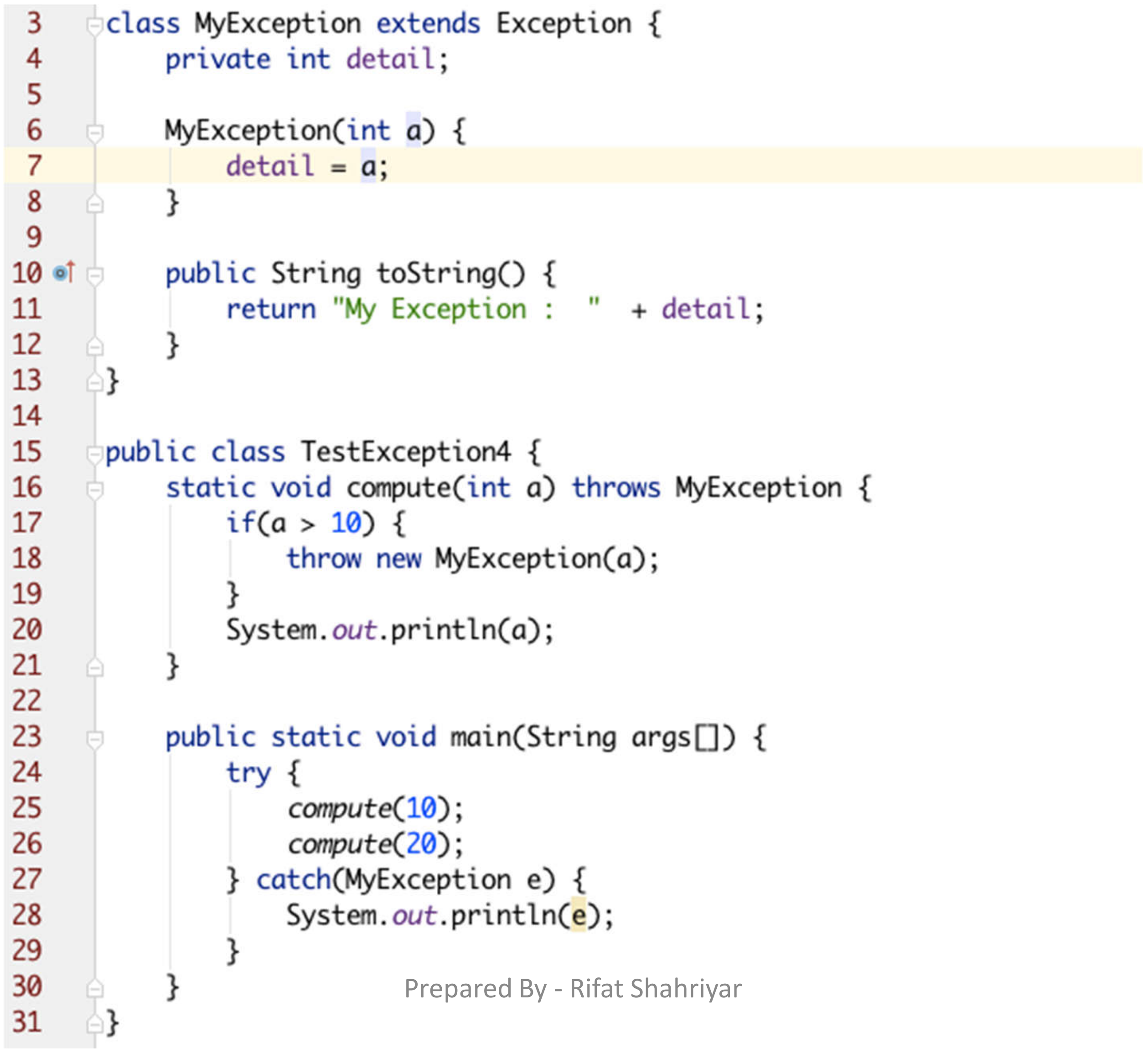
catch(ArithmeticException | NullPointerException e)

Throws



```java
public class TestException3
{
    public static void f() throws Exception {
        int a = 10;
        int b = 0;
        int c = a/b;
    }

    public static void main(String args[])
    {
        try {
            f();
        } catch(Exception e) {
            System.out.println (e);
            e.printStackTrace();
        }
        System.out.println("Hello World");
    }
}
```

Creating Custom Exceptions



```java
class MyException extends Exception {
    private int detail;

    MyException(int a) {
        detail = a;
    }

    public String toString() {
        return "My Exception :  " + detail;
    }
}

public class TestException4 {
    static void compute(int a) throws MyException {
        if(a > 10) {
            throw new MyException(a);
        }
        System.out.println(a);
    }

    public static void main(String args[]) {
        try {
            compute(10);
            compute(20);
        } catch(MyException e) {
            System.out.println(e);
        }
    }
}
```