What is the advantage of using exception handling?

The advantages of using exception handling: It enables a method to throw an exception to its caller. The caller can handle this exception. Without this capability, the called method itself must handle the exception or terminate the program. Often the called method does not know how to handle the exception. So it needs to pass the exception to its caller for handling.

Which of the following statements will throw an exception?

System.out.println(1 / 0);
System.out.println(1.0 / 0);
System.out.println(1 / 0); // Throws an exception
System.out.println(1.0 / 0); // Will not throw an exception

Point out the problem in the following code. Does the code throw any exceptions?

long value = Long.MAX_VALUE + 1;
System.out.println(value);

Adding 1 to Long.MAX_VALUE exceeds the maximum value allowed by a long value. But the current versions of Java does not report this as an exception.

What does the JVM do when an exception occurs? How do you catch an exception?

When an excepion occurs, Java searches for a handler in the catch clause. So to catch an exception in your program, you need to write a try-catch statement like this:

```
try {
}
catch (Exception ex) {
// Catch and process exception
}
```

What is the output of the following code?

```
public class Test {
public static void main(String[] args) {
try {
int value = 30;
if (value < 40)
throw new Exception("value is too small");
}
catch (Exception ex) {
System.out.println(ex.getMessage());
}
System.out.println("Continue after the catch block");
}
}
```

What would be the output if the line

int value = 30;

were changed to

int value = 50;

output is

value is too small
Continue after the catch block

The output would be if Line int value = 30; is changed to int value = 50; Continue after the catch block

Show the output of the following code.

(a)

```
public class Test {
public static void main(String[] args) {
for (int i = 0; i < 2; i++) {
System.out.print(i + " ");
try {
System.out.println(1 / 0);
}
catch (Exception ex) {
}
}
}
}
```

(b)

```
public class Test {
public static void main(String[] args) {
try {
for (int i = 0; i < 2; i++) {
System.out.print(i + " ");
System.out.println(1 / 0);
}
}
catch (Exception ex) {
}
}
}
```

a. 0 1

b. 0

Describe the Java Throwable class, its subclasses, and the types of exceptions.

See the section "Exceptions and Exception Types." The Throwable class is the root of Java exception classes. Error and Exception are subclasses of Throwable. Error describes fatal system errors, and Exception describes the errors that can be handled by Java programs. The subclasses of Error are LinkageError, VirtualMachineError, and AWTError. The subclasses of Exception include RuntimeException, IOException, AWTException, and InstantiationException.

What RuntimeException will the following programs throw, if any?

(a)

```
public class Test {
public static void main(String[] args) {
System.out.println(1 / 0);
}
}
```

(b)

```
public class Test {
public static void main(String[] args) {
int[] list = new int[5];
System.out.println(list[5]);
}
}
```
(c)
```
public class Test {
public static void main(String[] args) {
String s = "abc";
System.out.println(s.charAt(3));
}
}
```
(d)
```
public class Test {
public static void main(String[] args) {
Object o = new Object();
String d = (String)o;
}
}
```
(e)
```
public class Test {
public static void main(String[] args) {
Object o = null;
System.out.println(o.toString());
}
}
```
(f)
```
public class Test {
public static void main(String[] args) {
System.out.println(1.0 / 0);
}
}
```
a. ArithmeticException
b. ArrayIndexOutOfBoundsException
c. StringIndexOutOfBoundsException
d. ClassCastException
e. NullPointerException
f. No exception

What is the purpose of declaring exceptions? How do you declare an exception, and where? Can you
declare multiple exceptions in a method header?

The purpose of declaring exceptions is to tell the Java runtime system what can go wrong. You declare an exception using the throws keyword in the method declaration. You can declare multiple exceptions,

separated by commas.

What is a checked exception, and what is an unchecked exception?

A checked exception must be explicitly declared in the method declaration, if a method throws it. A checked exception must be caught in a try-catch block. An unchecked exception does not need to be declared and does not need to be caught. In Java, the only unchecked exceptions are RuntimeException and Error and their subclasses.

How do you throw an exception? Can you throw multiple exceptions in one throw statement?

You use the throw statement in the method to throw an exception. You cannot throw multiple exceptions in a single throw statement.

What is the keyword throw used for? What is the keyword throws used for?

throw is for throwing exceptions and throws is for claiming exceptions.

Suppose that statement2 causes an exception in the following try-catch block:

```
try {
statement1;
statement2;
statement3;
}
catch (Exception1 ex1) {
}
catch (Exception2 ex2) {
}
statement4;
```

Answer the following questions:

1. Will statement3 be executed?
2. If the exception is not caught, will statement4 be executed?
3. If the exception is caught in the catch block, will statement4 be executed?

1. No.
2. No.
3. Yes.

What is displayed when running the following program?

```
public class Test {
public static void main(String[] args) {
try {
int[] list = new int[10];
System.out.println("list[10] is " + list[10]);
}
catch (ArithmeticException ex) {
System.out.println("ArithmeticException");
}
catch (RuntimeException ex) {
System.out.println("RuntimeException");
}
catch (Exception ex) {
System.out.println("Exception");
```

```
}
}
}
```
RuntimeException
Reason: list[10] throws ArrayIndexOutOfBoundsException that is a subclass of RuntimeException.

What is displayed when running the following program?

```
public class Test {
public static void main(String[] args) {
try {
method();
System.out.println("After the method call");
}
catch (ArithmeticException ex) {
System.out.println("ArithmeticException");
}
catch (RuntimeException ex) {
System.out.println("RuntimeException");
}
catch (Exception e) {
System.out.println("Exception");
}
}
static void method() throws Exception {
System.out.println(1 / 0);
}
}
```
ArithmeticException
Reason: method() throws ArithmeticException.

What is displayed when running the following program?

```
public class Test {
public static void main(String[] args) {
try {
method();
System.out.println("After the method call");
}
catch (RuntimeException ex) {
System.out.println("RuntimeException in main");
}
catch (Exception ex) {
System.out.println("Exception in main");
}
}
static void method() throws Exception {
```

```
try {
String s ="abc";
System.out.println(s.charAt(3));
}
catch (RuntimeException ex) {
System.out.println("RuntimeException in method()");
}
catch (Exception ex) {
System.out.println("Exception in method()");
}
}
}
```

RuntimeException in method()
After the method call
Reason: s.charAt(3) throws StringIndexOutOfBoundsException that is a subclass of RuntimeException. This exception is caught in method(). The main method continues its normal execution flow.

What does the method getMessage() do?

the getMessage() is defined in the Throwable class to return a string that describes the exception.

What does the method printStackTrace() do?

To display trace information to the console.

Does the presence of a try-catch block impose overhead when no exception occurs?

No

Correct a compile error in the following code:

```
public void m(int value) {
if (value < 40)
throw new Exception("value is too small");
}
```

The method throws a checked exception. It must be caught or thrown. You may fix it as follows:

```
public void m(int value) throws Exception {
if (value < 40)
throw new Exception("value is too small");
}
```

Suppose you run the following code:

```
public static void main(String[] args) throws Exception2 {
m();
statement7;
}
public static void m() {
try {
statement1;
statement2;
statement3;
}
```

```
catch (Exception1 ex1) {
statement4;
}
finally {
statement5;
}
statement6;
}
```

answer the questions:

1. If no exception occurs, which statements are executed?
2. If statement2 throws an exception of type Exception1, which statements are executed?
3. If statement2 throws an exception of type Exception2, which statements are executed?
4. If statement2 throws an exception that is neither Exception1 nor Exception2, which statements are executed?

1. statement1, statement2, statement3, statement5, statement6, statement7.
2. statement1, statement2, statement4, statement5, statement6, statement7.
3. statement1, statement2, statement5.
4. statement1, statement2, statement5.

Suppose you run the following code:

```
public static void main(String[] args) {
try {
m();
statement7;
}
catch (Exception2 ex {
statement8;
}
}
public static void m() {
try {
statement1;
statement2;
statement3;
}
catch (Exception1 ex1) {
statement4;
}
finally {
statement5;
}
statement6;
}
```

answer the questions:

1. If no exception occurs, which statements are executed?

2. If statement2 throws an exception of type Exception1, which statements are executed?
3. If statement2 throws an exception of type Exception2, which statements are executed?
4. If statement2 throws an exception that is neither Exception1 nor Exception 2, which statements are executed?
1. statement1, statement2, statement3, statement5, statement6, statement7.
2. statement1, statement2, statement4, statement5, statement6, statement7.
3. statement1, statement2, statement5, statement8.
4. statement1, statement2, statement5.

The following method tests whether a string is a numeric string:

```
public static boolean isNumeric(String token) {
try {
Double.parseDouble(token);
return true;
}
catch (java.lang.NumberFormatException ex) {
return false;
}
}
```

Is it correct? Rewrite it without using exceptions.
Yes. It is correct. But a better way to write this method is using regular expression to test if the string is numeric.

Suppose that statement2 may cause an exception in the following code:

```
try {
statement1;
statement2;
statement3;
}
catch (Exception1 ex1) {
}
catch (Exception2 ex2) {
throw ex2;
}
finally {
statement4;
}
statement5;
```

Answer the following questions:
1. If no exception occurs, will statement4 be executed, and will statement5 be executed?
2. If the exception is of type Exception1, will statement4 be executed, and will statement5 be executed?
3. If the exception is of type Exception2, will statement4 be executed, and will statement5 be executed?
4. If the exception is not Exception1 nor Exception2, will statement4 be executed, and will statement5 be executed?
1. Yes to both.
2. Yes to both.

3. This exception is caught by the catch (Exception2 e2) clause and statement4 will be executed, but statement5 will not be executed because it is rethrown to its caller.
4. This exception is not caught. statement4 will be executed, but statement5 will not be executed.

What would be the output if line 16 is replaced by the following line?

throw new Exception("New info from method1");

The output will be

java.lang.Exception: New info from method1
at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:16)
at ChainedExceptionDemo.main(ChainedExceptionDemo.java:4)

How do you define a custom exception class?

To define a custom class, extend Exception or a subclass of Exception.