6. Exception Handling

6.1. Exceptions

An **exception** is a problem that occurs during program execution. Exceptions cause abnormal termination of the program.
Exception handling is a powerful mechanism that handles runtime errors to maintain normal application flow.

An exception can occur for many different reasons. Some examples:
- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications.
- Insufficient memory and other issues related to physical resources.

Note:

As you can see, exceptions are caused by user error, programmer error, or physical resource issues. However, a well-written program should handle all possible exceptions.

Q: Which of the following statements are true? [Select all that apply]

- ❏ Exception catching is a good practice
- ❏ We cannot catch any exceptions
- ❏ Exception catching improves program interface design
- ❏ If we don't catch exceptions, the program can shut down

Exception Handling

Exceptions can be caught using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception.

Syntax:

```
try {
        //some code
      } catch (Exception e) {
        //some code to handle errors
    }
```

A **catch** statement involves declaring the type of exception you are trying to catch. If an exceptionoccurs in the **try** block, the **catch** block that follows the **try** is checked. If the type of exception that occurred is listed in a **catch** block, the exception is passed to the **catch** block much as an argument is passed into a method parameter.

The **Exception** type can be used to catch all possible exceptions.

The example below demonstrates exception handling when trying to access an array index that does not exist:

```
public class MyClass {

        public static void main(String[ ] args) {

                try {

                  int a[ ] = new int[2];

                  System.out.println(a[5]);

                } catch (Exception e) {

                  System.out.println("An error occurred");

                }

        }

}
```

//Outputs "An error occurred"

Without the **try/catch** block this code should crash the program, as a[5] does not exist.

Note:

Notice the (**Exception e)** statement in the **catch** block - it is used to catch all possible Exceptions.

Q: Fill in the blanks to handle any possible exceptions.

```
int x = 12;
int y = 0;


_____ {
  int z = x / y;
  System.out.println(z);
}


_________ (Exception e) {
  System.out.println("Error");
}
```

6.2 Multiple Exceptions

throw

The **throw** keyword allows you to manually generate exceptions from your methods. Some of the numerous available exception types include the IndexOutOfBoundsException, IllegalArgumentException, ArithmeticException, and so on.

For example, we can throw an ArithmeticException in our method when the parameter is 0.

```
    int div(int a, int b) throws ArithmeticException {

            if(b == 0) {

              throw new ArithmeticException("Division by Zero");

            } else {

              return a / b;

            }

     }
```

The **throws** statement in the method definition defines the type of Exception(s) the method can throw.

Next, the **throw** keyword throws the corresponding exception, along with a custom message. If we call the **div** method with the second parameter equal to 0, it will throw an ArithmeticException with the message "Division by Zero".

Note:

Multiple exceptions can be defined in the throws statement using a **comma-separated**list.

Q: Fill in the blanks below to create a method that throws an IOException if the parameter is negative.

```
public void do(int x)
        _______  _________   {

         if(x<0)
         {


          _______  _________ IOException();
         }
}
```

Exception Handling

A single try block can contain multiple catch blocks that handle different exceptions separately.

Example:

```
try {
 //some code
} catch (ExceptionType1 e1) {
 //Catch block
} catch (ExceptionType2 e2) {
 //Catch block
} catch (ExceptionType3 e3) {
 //Catch block
}
```

Note:

All catch blocks should be ordered from most specific to most general.

Following the specific exceptions, you can use the **Exception** type to handle all other exceptions as the last catch.

Q: How many catch blocks can a try/catch block contain?

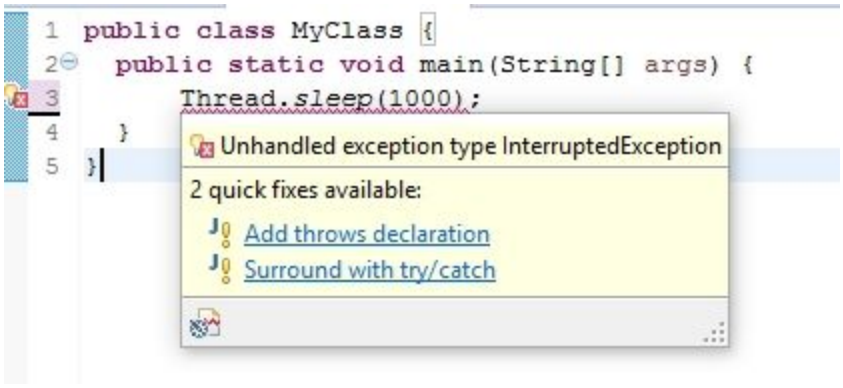
- Only two
- Only one
- None
- As many as you need

6.3 Runtime Vs Checked Exceptions

Types of Exceptions

There are two exception types, **checked** and **unchecked** (also called runtime). The main difference is that checked exceptions are checked when compiled, while unchecked exceptions are checked at runtime.

As mentioned in our previous lesson, Thread.sleep() throws an InterruptedException. This is an example of a **checked** exception. Your code will not compile until you've handled the exception.



```
public class MyClass {
        public static void main(String[ ] args) {
                try {
                  Thread.sleep(1000);
                } catch (InterruptedException e) {
                  //some code
                }
        }
}
```

We have seen examples of **unchecked** exceptions, which are checked at runtime, in previous lessons. Example (when attempting to divide by 0):

```
public class MyClass {
        public static void main(String[ ] args) {
          int value = 7;
          value = value / 0;
        }
}

/*
Exception in thread "main" java.lang.ArithmeticException: / by zero
 at MyClass.main(MyClass.java:4)
*/
```

Q: If not handled, which exception types prevent your program from compiling?

- Checked
- Runtime
- NullPointerException
- Both checked and runtime