Tuesday, October 18, 2022

Java Vettec 10032022

Exceptions Review

## What is an Exception?

Something unexpected that happens that interrupts the normal flow of the code instructions

### What are the 3 types of Exceptions?

1. errors – outside of your application something goes wrong – ex: OutOfMemoryError, VirtualMachineError

2. compile time exception = checked exception

3. runtime exception

examples - ArithmeticException

## What are the two categories of Exceptions?

1. checked

2. unchecked – remember errors and runtime exceptions fall under this category

## What is the purpose of Exceptions?

* to communicate to the users for what you can and can’t do with the code
* to communicate to the users what has gone wrong – as early as possible and as specifically as possible
* to help users identify the specific problem and what went wrong

## What is the purpose of handling Exceptions?

* to stop the program from breaking when something unexpected happens that your code isn’t written to handle

### When do you use “throw” vs “throws” keywords?

**throw –** when you are actively throwing an exception – in the method body itself

public int divide(int a, int b) {

if (b == 0)

**throw** new ArithmeticException();

}

* I can **throw** an exception without declaring it *if* it is a *runtime* exception that I’m throwing

**throws –** when you’re declaring that a method *could throw* an exception – in the method signature

public void myMethod() throws ArithmeticException { } – mainly for checked exceptions

* you can also declare unchecked exceptions
  + the compiler doesn’t care about, this would just be for the user
* purpose is so everytime you call the method you know you should put it in a try-catch block
  + to stop the code from compiling until you add a handler or declare it to warn your user

## How do you write a multi-catch block?

try {

} catch (SomeException | SomeOtherException | YetAnotherException e) {

}

## What order do multiple catch blocks need to be in?

* go from the MOST specific to the LEAST specific
  + because of the IS-A relationship between exceptions
  + when choosing a catch block, java goes in order top to bottom, and checks NOT the specific class but rather the IS-A relationship,
    - if the IS-A relationship test evaluates to true, then it stops looking through the catch blocks (so will never find a later more specific exception that matches)

try {

// some code AnotherException

// more code here – this will not get run

} catch (AnotherException e) { --- just like parameters in a method we give the TYPE and NAME of variable --- this could be ae or ex or myexception

// handle it somehow

// we can access all the properties and methods of the AnotherException class using the variable **e**

} catch (Exception e) {

//handle it somehow

}

// more code here – this will get run

## When do you need to import an Exception?

For convenience, the Java compiler automatically imports two entire packages for each source file:

(1) the java. lang package and

(2) the current package (the package for the current file).

If our exception is in another package like

* java.io
* java.sql

then you will need to import it at the top of the file

Exceptions – New Material

Exceptions and Inheritance –

Given the method:

public class Parent {

...

public void myMethod() throws IOException {

// some code here that might throw IOException

}

}

**Rule of Inheritance**

An overriding method must:

have the same or fewer exceptions that it throws

cannot have more exceptions or more general exceptions

which are valid method signatures for an overriding method in a child class?

\* More general exception specified

public class Child extends Parent {

...

public void myMethod throws Exception {

// some code here that might throw a Exception

}

}

\* More specific exceptions specified

\* Same exceptions specified

\* No exceptions specified

public class Child extends Parent {

...

public void myMethod throws FileNotFoundException {

// some code here that might throw a   
 // FileNotFoundException

}

}

public class Child extends Parent {

...

public void myMethod throws IOException {

// some code here that might throw IOException

}

}

public class Child extends Parent {

...

@Override

public void myMethod(){

// some code here

}

}











Exceptions with finally blocks –

**THE RULE IS THE FINALLY BLOCK ALWAYS RUNS**

Given the below code, what will be returned?

public int doSomething() {

try {

// some code that throws an exception

return 1; // THIS DOES NOT GET RUN

} catch (Exception e) {

return 2; // THIS RUNS AND TRIES TO RETURN 2

} finally {

return 3; // THIS ALWAYS RUNS – IT CHANGES THE   
 // RETURN VALUE TO 3

}

}

Return value is: 3  
 Why? **THE RULE IS THE FINALLY BLOCK ALWAYS RUNS**

Given the below code, what will the width of the rectangle be?

public Rectangle decreaseWidth() {

Rectangle rectangle = new Rectangle(1,1);

try {

rectangle.setWidth(rectangle.getWidth() – 1); // try to set width to 1 – 1 = 0

} catch (Exception e) {

return rectangle; // set width failed, so at this point width=1

throw MyException()

} finally {

rectangle.setWidth(0.00001); throw ItsException()

}

}

Width of rectangle now when the method ends: 0.00001  
 Why? **THE RULE IS THE FINALLY BLOCK ALWAYS RUNS**