

# Exceptions

When things go wrong

# Exceptions

- ▶ The term **exception** means an **exceptional condition** and is an occurrence that alters the normal program flow.
- ▶ What causes exceptions?
  - Hardware failures
  - resource exhaustion
  - bugs



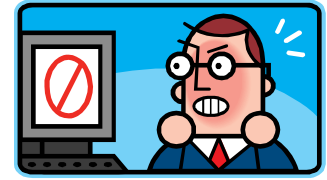
# Before Exceptions...



- ▶ In languages like C that don't have exceptions, programmers would make methods return a specific value if an error occurred, e.g.

```
int getPatientCountFromDatabase() {  
    if (!database.connect())  
        return -1;  
    else  
        return database.getPatientCount();  
}
```

# Why Is This Bad?



- ▶ It means the code which calls this method has to remember to check the return value
  - This code could be in a different library, or written by someone else
- ▶ What if 0 and -1 are legitimate return values, e.g.

```
int getPatientWeightChangeFromDatabase() {  
    if (!database.connect())  
        return ???;  
    else  
        return database.getPatientWeightChange();  
}
```

What should it return  
when an error occurs?

# Why Is This Bad?

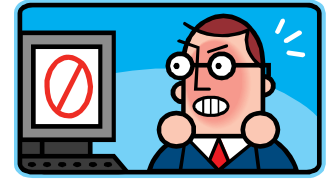


- ▶ It's hard to make sure that code recovers safely from an error, e.g.

```
boolean savePatientsToFile() {  
    file.open();  
  
    if (db.connect())  
        return false;  
  
    file.write(db.getPatients());  
  
    file.close();  
  
    return true;  
}
```

If an error occurs  
`file.close()` never gets  
called so its left open

# Why Is This Bad?



- ▶ It gets really complicated when the code that called the code that called the code needs to handle the error, e.g.

```
boolean amazingMethod() {  
    if (!notSoGoodMethod())  
        // Handle error!  
}  
  
boolean notSoGoodMethod() {  
    if (!methodWrittenByMonkeys())  
        return false;  
    // Do other stuff  
    return true;  
}  
  
boolean methodWrittenByMonkeys() {  
    // ERROR!!!!  
    return false;  
}
```

We have to keep checking  
the return values in each  
calling method

# Exceptions In Java



- ▶ Are built into the Java language
- ▶ Are NOT return values
- ▶ Use `try` and `catch` blocks, e.g.

```
try {  
    methodThatCouldFail();  
}  
catch (Exception ex) {  
    System.out.println("Error occurred:" + ex.toString());  
}
```

# Exception Terminology

- ▶ When an exception event occurs in Java, an exception is said to be “**thrown**”.
- ▶ The code responsible for doing something about the exception is called an “**exception handler**” and it “**catches**” the thrown exception.
- ▶ For things that must happen whether or not there is an exception, there is the “**finally**” clause.



# Exception Objects

- ▶ Exceptions are objects of class `Exception`, e.g.

```
throw new Exception("Could not connect to DB")
```

- ▶ It has several useful methods
  - `getMessage()` – returns the error message
  - `printStackTrace()` – prints the stack trace to the console...

```
java.lang.NullPointerException  
    at MyClass.bar(MyClass.java:9)  
    at MyClass.foo(MyClass.java:6)  
    at MyClass.main(MyClass.java:3)
```


Tells us what called  
what before the  
exception occurred

# Example




```
int getCountFromDatabase() throws Exception {  
    if (!database.connect())  
        throw new Exception("Could not connect to DB");  
}
```

```
void printCount() {  
    try {  
        int count = getCountFromDatabase();  
  
        System.out.println("Count:" + count);  
    }  
    catch (Exception ex) {  
        System.out.println("Error occurred:" + ex.toString());  
    } finally {  
        connection.close();  
    }  
}
```



If no exception,  
then try block  
continues



If exception occurs,  
then we jump to  
the catch block

Finally always  
executes

# Finally...



```
void saveDataToFile() {  
  
    try {  
        file.open();  
  
        file.write(data);  
  
        file.close();  
    }  
    catch (Exception ex)  
    {  
        log("File Error");  
        file.close();  
    }  
}
```

- ▶ Sometimes we need to execute some code regardless of whether an exception is thrown
- ▶ For example a resource like a file or database connection may need to be released

Code is duplicated!

A green rounded rectangle with a double border containing the text "Code is duplicated!". Two green arrows originate from this box: one points to the `file.close();` line inside the `try` block, and the other points to the `file.close();` line inside the `catch` block, illustrating that the same resource management code is repeated in two different locations.

# Finally...



```
void saveDataToFile() {  
  
    try {  
        file.open();  
  
        file.write(data);  
    }  
    catch (Exception ex) {  
        log("File Error");  
    }  
    finally {  
        file.close();  
    }  
}
```

- ▶ Code in the `finally` block is called
  - After the `try` if no exception occurred
  - After the `catch` if an exception did occur
- ▶ Why is this necessary? Couldn't the code just go at the end of the method...

# Finally...



```
void saveDataToFile() {  
  
    try {  
        file.open();  
  
        file.write(data);  
    }  
    catch (Exception ex) {  
        log("File Error");  
        return;  
    }  
    finally {  
        file.close();  
    }  
}
```

- ▶ Code in the `finally` block is even called if a catch block has a `return` statement

Still called even  
though `catch`  
block returns

```
public class ExampleExceptions {

    public static void main(String[] args) {
        myMethod();
    }

    static void myMethod() {

        try{
            // do stuff
            System.out.println("inside try.");

        } catch(Exception e) {
            // do exception handling
            System.out.println("inside catch.");
        } finally {
            // do cleanup
            System.out.println("inside finally.");
        }

    }

}
```

**REVIEW-- Why is the method myMethod static?**

# Student Practice

- ▶ Run the program on the previous slide.
  - Why doesn't "inside catch." print out to the console?
- ▶ Add the following in the **try** and run it again:

```
int x = 8/0;
```

- ▶ Add the following in the **catch** one at a time and re-run the program:

```
System.err.println(e.toString());  
System.err.println(e.getMessage());  
e.printStackTrace();
```

# “Try” It Out!

- ▶ What happens if you have no finally?
- ▶ What happens when you have a try by itself?
- ▶ What happens when you have a try and then some code and then the catch? A try and then a catch and then some code and then the finally?
- ▶ What happens if you have try and finally but no catch?



# Throwing Exceptions

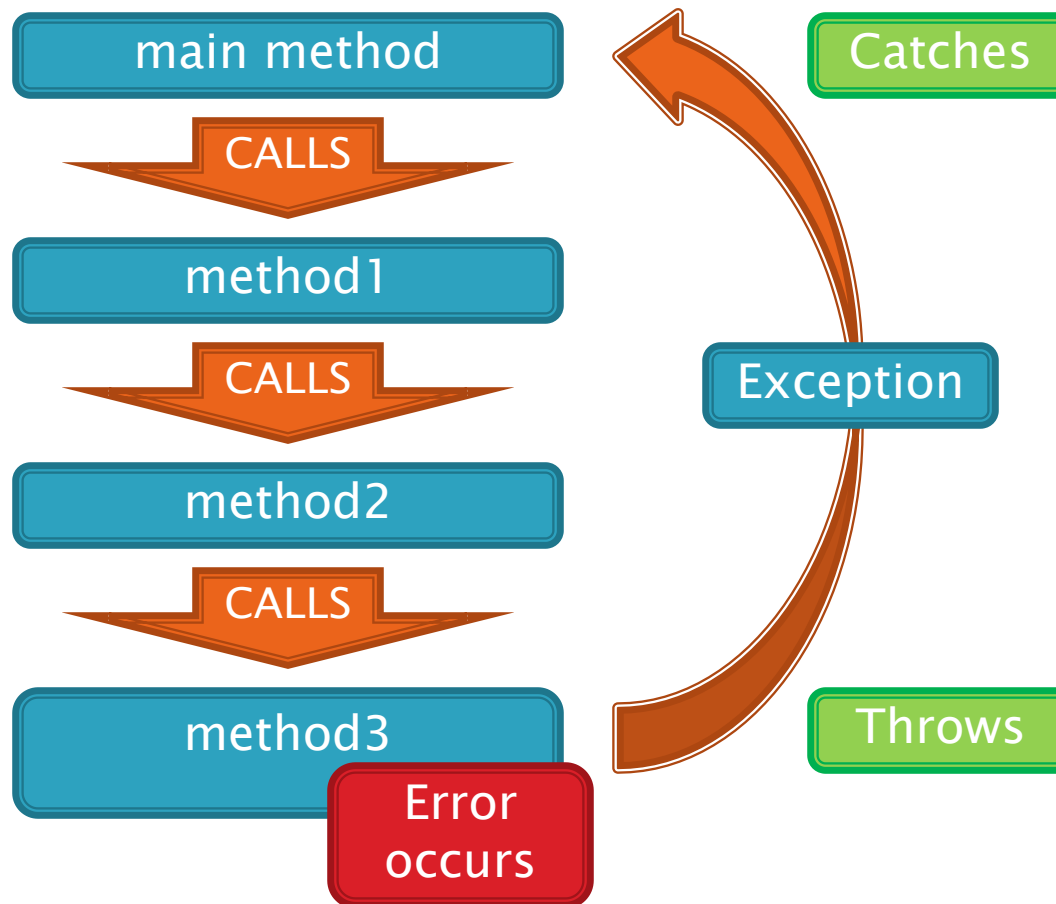


- ▶ If a method throws an exception and doesn't handle it (i.e. there is no `catch` block) then it must declare that it `throws` an exception, e.g

```
int getCountFromDatabase() throws Exception {  
    if (!database.connect())  
        throw new Exception("Could not connect to DB");  
}
```

- ▶ Exceptions can be constructed with a `String` which is the error message
- ▶ This can be retrieved in the `catch` block using `getMessage()`

# Try, throw, catch



# Ducking Exceptions

```
try{  
    // call method that throws an exception  
    String myAnswer = doSomething("test");  
} catch(Exception e) {  
    // handle exception if it occurs  
}  
finally {  
    // cleanup  
}
```

```
static String doSomething(String s) throws Exception {
```

```
    if(POTENTIAL PROBLEM) {  
        throw new Exception();  
    }
```

```
    // BODY OF METHOD
```

```
    return s;
```

```
}
```



# Hands On



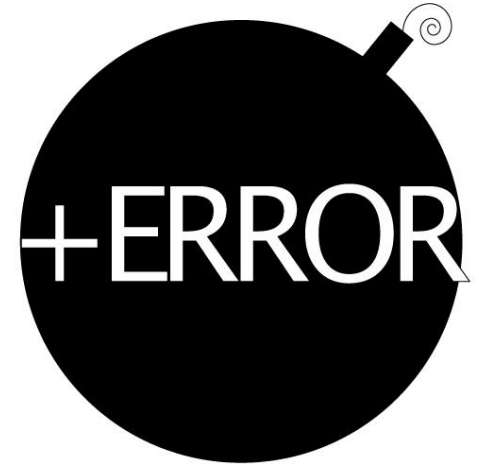
- ▶ Exceptions are Objects
  - `java.lang.Exception`
- ▶ Look up the Java API SE 5.0 (1.5)
  - What is the superclass of Exception?
  - What is the sibling of Exception?
  - Do you see the subclass called RuntimeException?

# Checked vs. Unchecked Exceptions

- ▶ Checked by the Compiler – Handle or Declare
  - Exceptions Other Than Runtime Exceptions
  - Your Own Custom Exceptions
- ▶ Unchecked – Can Handle If You Want
  - Runtime Exceptions
  - Errors



# Errors



- ▶ Exceptional situation that aren't programmatic
- ▶ Not required to handle these
- ▶ Example--JVM running out of memory

```
try{  
    // do stuff  
    System.out.println("inside try.");  
  
} catch(Error e) {  
    // do exception handling  
}
```

# Runtime Exceptions

- ▶ These are a special kind of exception which aren't checked by the compiler
- ▶ They extend the `RuntimeException` class
- ▶ They can usually be handled by fixing programming logic, e.g.
  - `NullPointerException`
  - `ArrayIndexOutOfBoundsException`
  - `DivideByZeroException`
  - `ClassCastException`

# Runtime Exceptions

- ▶ Thus it's usually bad practice to use these exceptions with `try` and `catch`, e.g.

```
void calculateAverageAge(int totalAge, int numPeople) {  
  
    try {  
        int avgAge = totalAge / numPeople;  
    }  
    catch (DivideByZeroException ex) {  
    }  
}
```

Would be better to write the method so that it checks that `numPeople > 0` before doing the division



# The Stack Trace



- ▶ This will be displayed if you don't handle a runtime exception or you call `printStackTrace()` on an exception object
- ▶ It will help you determine where the error occurred in your code
- ▶ It looks ugly but it can be very helpful!

# Example



Each item has a  
source file and  
line number

```
org.openmrs.api.APIAuthenticationException: Privileges required: [Manage Scheduler]
  at org.openmrs.aop.AuthorizationAdvice.throwUnauthorized (AuthorizationAdvice.java:115)
  at org.openmrs.aop.AuthorizationAdvice.before (AuthorizationAdvice.java:94)
  at org.springframework.aop...MethodBeforeAdviceInterceptor.invoke (MethodBeforeAdviceInterceptor.java:49)
  at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed (ReflectiveMethodInvocation.java:171)
  at org.springframework.aop.framework.JdkDynamicAopProxy.invoke (JdkDynamicAopProxy.java:204)
  at $Proxy208.getTaskByName (Unknown Source)
  at sun.reflect.NativeMethodAccessorImpl.invoke0 (Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke (Unknown Source)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke (Unknown Source)
  ...
  at sun.reflect.DelegatingMethodAccessorImpl.invoke (Unknown Source)
  at java.lang.reflect.Method.invoke (Unknown Source)
  at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection (AopUtils.java:307)
  at org.springframework.aop.framework.JdkDynamicAopProxy.invoke (JdkDynamicAopProxy.java:198)
  at $Proxy228.getTaskByName (Unknown Source)
  at org.openmrs.module.usagestatistics.ModuleActivator.unregisterAggregationTask (ModuleActivator.java:115)
  at org.openmrs.module.usagestatistics.ModuleActivator.shutdown (ModuleActivator.java:63)
  at org.openmrs.module.ModuleFactory.stopModule (ModuleFactory.java:740)
```

If the exception occurred  
in a library, then look for  
the first reference to a  
class in your project

# Hands On



```
public class RuntimeExceptionExample {  
  
    public static void main(String[] args) {  
  
        System.out.println("hello from main");  
  
        myMethod();  
    }  
  
    static void myMethod() {  
        System.out.println("hello from my method.");  
  
        myNextMethod();  
    }  
  
    static void myNextMethod() {  
        int x = 8/0;  
    }  
}
```

What happens when you run this??

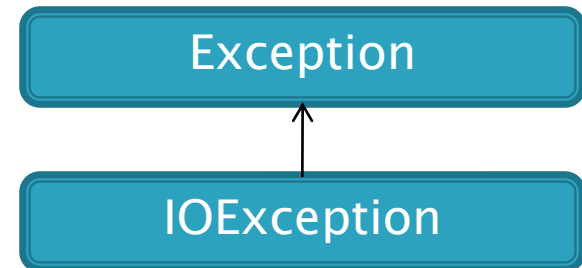
# Exception Matching

```
void saveDataToRemoteFile() {  
  
    try {  
        file.open();  
  
        file.write(data);  
    }  
    catch (IOException ex) {  
        log("File Error");  
    }  
    catch (SocketException ex) {  
        log("Connection Error");  
    }  
    finally {  
        file.close();  
    }  
}
```

- ▶ Code in a `try` block might throw different exceptions
- ▶ These can be caught in separate catch blocks

# Exception Hierarchy

```
void saveToRemoteFile() {  
  
    try {  
        file.open();  
  
        file.write(data);  
    }  
    catch (IOException ex) {  
        log("File Error");  
    }  
    catch (Exception ex) {  
        log("Unknown Error");  
    }  
    finally {  
        file.close();  
    }  
}
```



- ▶ The first catch block one with a matching exception type will be used
- ▶ Exception is the super class of all exceptions so will match any exception

# Exception Hierarchy

```
void saveToRemoteFile() {  
  
    try {  
        file.open();  
  
        file.write(data);  
    }  
    catch (Exception ex) {  
        log("Unknown Error");  
    }  
    catch (IOException ex) {  
        log("File Error");  
    }  
    finally {  
        file.close();  
    }  
}
```

- ▶ So **catch blocks** should be put in the order of the exception class hierarchy

This will catch all exceptions so the second block will never be used

# Create Your Own Exceptions

```
class myException extends Exception { }

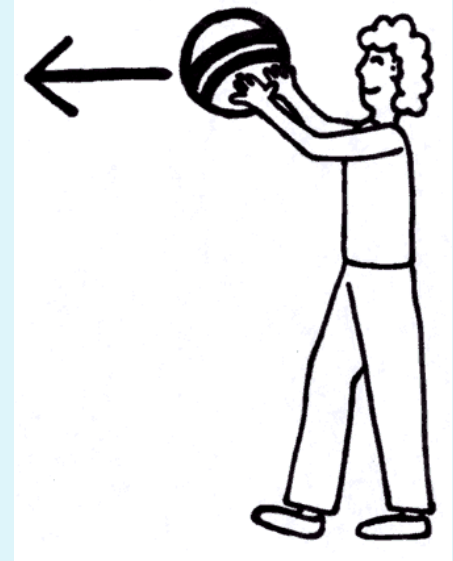
class TextEx {
    void doStuff() throws myException {
        throw new myException();
    }
}
```

What is wrong with the above?

Will it compile?

# Rethrowing Exceptions

```
void someMethod() {  
    doStuff();  
}  
  
void doStuff() throws Exception {  
    try {  
        throw new Exception();  
    } catch (Exception e) {  
        throw e;  
    }  
}
```



What happens when you run this?



# Exception Types

- ▶ JVM Exceptions
  - Thrown by the JVM
- ▶ Programmatic Exceptions
  - Thrown by the application or API programmers



# Null Pointer Exception Example

```
public static void main(String[] args) {  
    String s = null;  
  
    if(s.equals("hi")) {  
        // do something  
    }  
}
```

# Number Format Exception Example

```
public static void main(String[] args) {  
  
    int answer = divideLargeNumbers(5, 6);  
    System.out.println(answer);  
}  
  
static int divideLargeNumbers(int i, int j) {  
  
    if(j < 100 || i < 100) {  
        throw new NumberFormatException();  
    }  
  
    return i/j;  
}
```

# Common Exceptions By Type

## JVM

- ▶ `ArrayIndexOutOfBoundsException`
- ▶ `ClassCastException`
- ▶ `NullPointerException`
- ▶ `ExceptionInInitializerError`
- ▶ `StackOverflowError`
- ▶ `NoClassDefFoundError`

## Programmatic

- ▶ `IllegalArgumentException`
- ▶ `IllegalStateException`
- ▶ `NumberFormatException`
- ▶ `AssertionError`

# References

- ▶ Sun's Java tutorials:  
<http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html>