Today’s topic:

- Exceptions and enumerations.
Runtime Errors

- All of us have experienced syntax errors. This is an error in the code that causes that prevents the compiler from compiling the code.

- Some errors are only discovered during runtime. In Java these runtime errors come in two types: Errors and Exceptions.

- In Java Errors and Exceptions are just objects that are subclasses of the Throwable class.


Error Class

- A Java Error “indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.” (From Java API).

- Typically you cannot do anything to handle Errors.

Exception Class

- A Java Exception “and its subclasses ... indicates conditions that a reasonable application might want to catch.” (From Java API).

- Instead of having your program crash and exit, you may choose to try and handle the exception without stopping the program.


An Example

```java
public class ExceptionApplication {
    public static void main(String[] args) {
        int myNums[] = {1, 2, 3};
        System.out.println("Starting run...");
        System.out.println(myNums[0]);
        System.out.println(myNums[1]);
        System.out.println(myNums[2]);
        System.out.println(myNums[3]);
        System.out.println("...ending run.");
    }
}
```

```
<terminated> ExceptionApplication [J] java.Application C:\Program Files\Java\jre7\bin\java.exe
Starting run...
1
2
3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
    at ExceptionApplication.main(ExceptionApplication.java:16)
```
Types of Exceptions

- There are of course different types of exceptions.
- The exception classes utilize inheritance and the *is-a* relationship.
- For example and IndexOutOfBoundsException *is-a* RuntimeException

How to handle Exceptions

- Use `try` and `catch`
- In this example code I’m catching *any* Exception, but I could catch a specific Exception type.

```java
try {
    //Code that may throw exception.
}

catch(Exception e) {
    //Code to run when the exception is caught.
}
```
An Example

- I’m only going to catch the ArithmeticException.

```java
public class ExceptionApplication {
    public static void main(String[] args) {
        System.out.println("Starting run...");
        try {
            System.out.println(divideBy(Integer.valueOf(args[0])));
        } catch (ArithmeticException e) {
            System.out.println("I can't do the impossible!");
        }
        System.out.println("...ending run.");
    }
    public static int divideBy(int denominator) {
        return 10 / denominator;
    }
}
```

Finally clause

- Sometimes there is additional code you wish to execute whether the exception is caught or not. This can be added in a `finally` block.

```java
try{
    System.out.println(divideBy(Integer.valueOf(args[0])));
} catch (Exception e){
    System.out.println("I can't do the impossible!");
}
finally{
    System.out.println("...ending run.");
}
```
Playing Catch?

- You can think of exceptions as a sports metaphor.

- Think of baseball, football or rugby.
  - One player (method) can catch the ball (exception).
  - They may hold on to it (handle the exception).
  - Or
  - They may throw it on to someone else (the calling method).

Throwing Exceptions

- Any method that may throw an exception, needs to declare it in the method declarations:

```
public void someMethod() throws SomeException
```

- This is visible in the API so other programmers know they need to handle a potential Exception.
Catching and Throwing Exceptions.

- Sometimes we will simply catch an exception and throw it again. Why?

  We may not know the best way to handle it under the circumstances. We don’t know the details of who or why the method was called.

  For example:
  - Drawing the top card from an empty deck, should cause an exception.
  - That exception can be handled best by the program that sent the drawCard message.

  What are some ways to handle this OutOfCardsException?
  - Maybe it signals the end of the game.
  - Maybe it means to reset the deck and start over.
  - Maybe it means to take a discard pile and reshuffle it into a new deck.

Enumerations in Java

- Enumerations allow you to define a set group of items.

- Example:

  ```java
  public enum Month {
    January, February, March, April, May, June, July,
    August, September, October, November, December
  }
  ```
Benefits of Enumerations

- An enumeration is will define the legal values for that type of enumeration and simplifies validation.

- Enumerations can be used inside a case statement in Java.

- The ordinal value can be used for ordering the list.

- You can easily add additional constants to the enumeration without affecting existing code.