Exceptions

CS110: Introduction to Computer Science

Exceptions

- Sometimes good code goes bad.
- For example let's type "Hello" into the following code

```java
import java.util.Scanner;

public class Starter {
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        System.out.println("Please enter an integer");
        int value = scan.nextInt();
        scan.close();
    }
}
```

Output

```
Please enter an integer
hello
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at Starter.main(Starter.java:7)
```

Exceptions

- Exceptions are a part of java that let us know something is gone wrong.
- Unlike just crashing, exceptions are something we can handle.
- Sometimes, the code will crash but sometimes this is what we want to happen.

How to Handle exception

- We use a Try-Catch Block

```
try {
    System.out.println("Please enter an integer");
    value = scan.nextInt();
} catch(InputMismatchException e){
    System.out.println("You did not enter an integer.");
}
```

How to Handle exception

InputMismatchException is the **Exception** Class that is **"thrown"** by **Scanner**

- We use a Try-Catch Block

```
try {
    System.out.println("Please enter an integer");
    valu[illegible]tInt();
} catch (InputMismatchException e)
    System.out.println("You did not enter an integer.");
}
```

How to Handle exception

Let's try and make this better

- We use a Try-Catch Block

```
try {
    System.out.println("Please enter an integer");
    value = scan.nextInt();
} catch(InputMismatchException e){
    System.out.println("You did not enter an integer.");
}
```

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class Starter {
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        boolean valueEntered = false;
        int value = 0;
        while (!valueEntered){
            try {
                System.out.println("Please enter an integer");
                value = scan.nextInt();
                valueEntered = true;
            } catch(InputMismatchException e){
                System.out.println("You did not enter an integer.");
                scan.nextLine();
            }
        }
        System.out.println("Good Job: "+value);
        scan.close();
    }
}
```

Making our Own Exceptions

- We can make our own exception classes. It's easy.
- Just extend the `Exception` class.
- This is literally it

```java
public class InvalidDenominatorException extends Exception {

}
```

Using our Exception

- Now that we've made a class, we can use it when something goes wrong.
- To "use" an exception we need to `throw` it.
- Also, any function that can "throw" an exception must specify in its signature that it may throw.

```java
public class Fraction {
    private int num, den;

    public Fraction(int num, int den) throws InvalidDenominatorException {
        if(den<=0) {
            throw new InvalidDenominatorException();
        }
        this.num = num;
        this.den = den;
    }

    public void setDen(int den) throws InvalidDenominatorException {
        if(den<=0) {
            throw new InvalidDenominatorException();
        }
        this.den = den;
    }

    public void setNum(int num){
        this.num = num;
    }

    public String toString(){
        return num+"/"+den;
    }
}
```

```java
public class Fraction {
    private int num, den;

    public Fraction(int num, int den) throws InvalidDenominatorException {
        if(den<=0) {
            throw new InvalidDenominatorException();
        }
        this.num = num;
        this.den = den;
    }

    public void setDen(int den) throws InvalidDenominatorException {
        if(den<=0) {
            throw new InvalidDenominatorException();
        }
        this.den = den;
    }

    public void setNum(int num){
        this.num = num;
    }

    public String toString(){
        return num+"/"+den;
    }
}
```

Functions and constructors can throw exceptions.

Testing Exceptions

- We should test Exceptions, just like we test everything else.
- In this case our JUnit tests should make sure that an exception is thrown was bad data is passed.

```
@Test
void testInvalidDen() {
    try {
        Fraction f = new Fraction(1, 0);
    } catch (InvalidDenominatorException e){
        return;
    }
    //We were able to create the fraction,
    //Which is bad
    fail();
}
```