Exceptions

CS110

Austin College

What’s the problem with this code?

```java
public class Fraction {

  private int num;
  private int den;

  public Fraction(int n, int d){
    num = n;
    den = d;
    if (den == 0){
      den = 1;
      num = 0;
    }
  }

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

Errors

- The person using the fraction class may have meant to create the fraction

```
Fraction frac = new Fraction(33,9);
```

- But accidentally typed

```
Fraction frac = new Fraction(33,0);
```

- The meaning is fundamentally changed by this small typo and can cause hours lost trying to track down.
- We could print an error message but…
- If they are working in the GUI only they might not even read the error messages at the bottom.

Exceptions

- **Exceptions provide an "alternative" way to return a message**
- **When a program encounters an exception, it has two options**
  - Catch and handle the exception
  - Intentionally crash the program
- **Intentionally crashing is not as bad as it sounds. Many times, it’s better to crash**
  - For example, data loss

Throwing an Exception

- **Any function (including constructors) can throw an exception.**
- **Two parts to throwing an exception**
  - Modifying the method header to specify the type of exception we are throwing
  - Actually throwing the exception

```java
public class Fraction {

  private int num;
  private int den;

  public Fraction(int n, int d) throws Exception{
    num = n;
    den = d;
    if (den == 0){
      throw new Exception();
    }
  }

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

Why `new Exception()`

- **Exception is a class built into Java**
- **When you are throwing Exception, you are actually throwing an Object that is subclassed from Exception.**
- **Wait… subclassed… does that mean?**
- **Yes, you can build your own Exception types.**
- **In fact, you shouldn’t throw "Exception" you should throw a subclass.**
  - Exception is too vauge could be anything….

Subclassing Exception

- Subclasses of Exceptions can be really boring.
- They convey meaning just by being a different class.

```
public class DivideByZero extends Exception {

  public DivideByZero() {
    super();

  }

  public DivideByZero(String message) {
    super(message);

  }

}
```

Cleaning up Old code

- Let’s go back and clean up Fraction

```
public class Fraction {

  private int num;
  private int den;

  public Fraction(int n, int d) throws DivideByZero{
    num = n;
    den = d;
    if (den == 0){
      throw new DivideByZero();
    }
  }

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

Cleaning up Old code

- If we want to, we can even pass a message

```java
public class Fraction {

  private int num;
  private int den;

  public Fraction(int n, int d) throws DivideByZero{
    num = n;
    den = d;
    if (den == 0){
      throw new DivideByZero("Den can’t be zero");
    }
  }

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

Handeling an exception

- **We handle an exception by using a try…catch statement.**
- **Java "tries" to run the code in the try block, if it encounters an exception, then it "catches" it.**
- **The Catch then tries to fix the problem or it throws an exception letting the code that called it can't be fixed**
- **A catch will only "catch" its specified type of exception.**
- **Any uncaught exception will cause the program to crash.**
  - Not terrible because we will know exactly where we crash and the cause of it.

Try...Catch

```java
public static void main(String[] args) {
  Fraction frac;

  try {
    frac = new Fraction(2,0);
    System.out.println("This will never print");
  } catch (DivideByZero e){
    frac = null;
  }

  if (frac!=null){
    System.out.println(frac.toString());
  } else {
    System.out.println("Fraction doesn't exist");
  }
}
```

Try…Catch

We need to **declare** frac before the try…catch if we want to use it after (this is a scoping issue)

```
public static void main(String[] args) {
  Fraction frac;

  try {
    frac = new Fraction(2,0);
  } catch (DivideByZero e){
    frac = null;
  }

  if (frac!=null){
    System.out.println(frac.toString());
  } else {
    System.out.println("Fraction doesn't exist");
  }
}
```

```java
public static void main(String[] args) {
  Scanner scan = new Scanner(System.in);
  Fraction frac = null;
  boolean invalid;
  do{
    invalid = false;
    System.out.println("Please enter a num and den. Den canot be zero:");
    int num = scan.nextInt();
    int den = scan.nextInt();
    try {
      frac = new Fraction(num,den);
    } catch (DivideByZero e){
      invalid = true;
      System.out.println(e.getMessage());
    }
  } while (invalid);
  System.out.println("Fraction you entered is "+frac.toString());

}
```

Bit of guidance

- **Don't throw or catch the `Exception` class**
- **Subclassing is easy and `Exception` provides no information**
- **Catching `Exception` is even worse.**
  - You might catch exceptions that you didn't intend to
  - Those exceptions could stop you from doing something worse than crashing.
  - Seen this in practice and it makes my eyes bleed.