Lecture 23

Exception handling and a bit of processing
Exceptions

• Let’s run a little test using an ArrayList from last class

```java
import java.util.*;

public class ArrayList_InClass {

    public static void main(String[] args) {
        ArrayList<String> cities = new ArrayList<>();

        cities.add("Madrid");
        cities.add("New York");
        cities.add("Sydney");
        cities.add("Seoul");

        System.out.println(cities.get(4));
    }
}
```
Exceptions

• Error! You’ve probably seen this kind of thing before

Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 4, Size: 4
  at java.util.ArrayList.rangeCheck(ArrayList.java:653)
  at java.util.ArrayList.get(ArrayList.java:429)
  at ArrayList_InClass.main(ArrayList_InClass.java:13)
Exceptions

• That’s a runtime error

• That means java hit something in our code that it couldn’t process

• runtime errors *throw* exceptions

• an exception is an object that represents the error
Exceptions

• If exceptions aren’t handled, the program breaks

• THAT’S…..BAD

• So, we deal with them so our program will keep running
import java.util.*;
import java.util.Scanner;

public class Exceptions {

    public static void main(String[] args) {
        int number1, number2;
        Scanner input = new Scanner(System.in);

        System.out.println("Please enter 2 numbers: ");
        number1 = input.nextInt();
        number2 = input.nextInt();

        System.out.println(number1 / number2);
    }
}

Exceptions example
import java.util.*;
import java.util.Scanner;

public class Exceptions {

    public static void main(String[] args) {
        int number1, number2;
        Scanner input = new Scanner(System.in);

        System.out.println("Please enter 2 numbers: ");
        number1 = input.nextInt();
        number2 = input.nextInt();

        int divideResult = result(number1, number2);
        System.out.println(divideResult);
    }

    public static int result(int num1, int num2) {
        if (num2 == 0) {
            System.out.println("Can't divide by zero");
            System.exit(0); // Quit the program! Not great.
        }

        return num1 / num2;
    }
}
Exceptions

- The method shouldn’t decide whether or not to quit the program, the caller should decide (like the main method)

- So how can a method let the caller know there was a problem?
try...catch

• We can *throw* an exception in the method and *catch* it in the caller to make sure our program doesn’t crash.
import java.util.*;

public class Exceptions {

    public static void main(String[] args) {
        int number1, number2;
        Scanner input = new Scanner(System.in);

        System.out.println("Please enter 2 numbers: ");
        number1 = input.nextInt();
        number2 = input.nextInt();

        try {
            try {
                int divideResult = result(number1, number2);
                System.out.println(divideResult);
            } catch (ArithmeticException e) {
                System.out.println("An int can't be divided by zero");
            }

            System.out.println("And we keep going...");
        }

        public static int result(int num1, int num2) {
            if (num2 == 0) {
                throw new ArithmeticException("Divisor cannot be zero");
            }
            return num1 / num2;
        }
    }
}
try...catch

- The ArithmeticException thrown in that example is an object created from extending the Exception class
try...catch

**Listing 12.5  InputMismatchExceptionDemo.java**

```java
import java.util.*;

public class InputMismatchExceptionDemo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        boolean continueInput = true;

        do {
            try {
                System.out.print("Enter an integer: ");
                int number = input.nextInt();
                // Display the result
                System.out.println("The number entered is " + number);
                continueInput = false;
            } catch (InputMismatchException ex) {
                System.out.println("Try again. (" + "Incorrect input: an integer is required)");
                input.nextLine(); // Discard input
            }
        } while (continueInput);
    }
}
```
Exception types

- Object
  - Throwable
    - Exception
      - ClassNotFoundException
      - IOException
      - RuntimeException
        - Many more classes
      - LinkageError
    - Error
      - VirtualMachineError
        - Many more classes
      - Many more classes
    - ArithmeticException
      - NullPointerException
      - IndexOutOfBoundsException
      - IllegalArgumentException
      - Many more classes
Checked vs unchecked

• RuntimeException, Error and their subclasses are known as *unchecked* exceptions

• All the others are checked

• Checked means that java makes you use try-catch blocks before compiling

• Unchecked exceptions can happen anywhere, and are usually logic errors in the program that need to be fixed (like NullPointerExceptions, IndexOutOfBoundsException, etc)
Checked vs unchecked

- All methods must say what kinds of checked exceptions they might throw

- You don’t have to for Error and RuntimeException (since they are unchecked)
Declaring an exception

• Use the `throws` keyword in the method header

```java
public void myMethod() throws IOException
```
public class CircleWithExceptions {
    private double radius;

    public CircleWithExceptions(double radius){
        setRadius(radius);
    }

    public void setRadius(double radius) throws IllegalArgumentException{
        if (radius >= 0){
            this.radius = radius;
        } else {
            throw new IllegalArgumentException("Radius must be greater than zero");
        }
    }
}
public class TestCircle_prep {

    public static void main(String[] args) {

        try {
            CircleWithExceptions circle1 = new CircleWithExceptions(1.0);
            CircleWithExceptions circle2 = new CircleWithExceptions(-1.0);
            CircleWithExceptions circle3 = new CircleWithExceptions(4.0);
        }
        catch (IllegalArgumentException e){
            System.out.println(e.getMessage());
        }
    }
}
Information in exceptions

• The information held in an exception can be very helpful in figuring out what went wrong and why
Information in exceptions

java.lang.Throwable

+getMessage(): String
Returns the message that describes this exception object.

+toString(): String
Returns the concatenation of three strings: (1) the full name of the exception class; (2) "::" (a colon and a space); (3) the getMessage() method.

+printStackTrace(): void
Prints the Throwable object and its call stack trace information on the console.

+StackTraceElement[]
Returns an array of stack trace elements representing the stack trace pertaining to this exception object.
Finally

• There is also another block you can use after the try-catch called finally

• This block will always execute no matter if an exception is thrown or not
Finally

```java
try {
    statements;
}
catch (TheException e) {
    handling e;
}
finally {
    finalStatements;
}
```
Finally

• This is useful for cleanup stuff like closing files
So when would we use exceptions?

- Generally, if it’s a very specific exception you’re handling and you can just take care of it in the method, do that.

- If you want the caller to take care of it (like the main method), throw an exception

- If it’s an exception that might happen multiple places in your code, then it’s probably a good candidate for an exception class
So when would we use exceptions?

```java
try {
    System.out.println(refVar.toString());
}
catch (NullPointerException ex) {
    System.out.println("refVar is null");
}

is better replaced by

if (refVar != null)
    System.out.println(refVar.toString());
else
    System.out.println("refVar is null");
```
So when would we use exceptions?

- Don’t abuse try-catches to take care of simple stuff
- Do use them to catch errors so your program doesn’t crash!
And now for something completely different...
Processing - drawing encapsulation

- Let’s take a look at our Pong example
Processing - drawing encapsulation

- In our game, we’re drawing the paddle in the main game loop by grabbing all the data from our paddle object and drawing a rectangle.

- What if we have our paddle object take care of the drawing as well? That would be much cleaner.
void draw() {
    rect(xPos, yPos, width, height);
}

• But our paddle doesn’t know anything about the rect() method!

• That method is defined in the PApplet class

• So how can we get that into our Paddle object?
Processing - drawing encapsulation

...     PApplet canvas;

    Paddle(int gameWidth, int gameHeight, PApplet canvas){
        this.gameWidth = gameWidth;
        this.gameHeight = gameHeight;
        xPos = gameWidth / 2;
        yPos = gameHeight - height;

        this.canvas = canvas;
    }

    ...
• All the drawing is taken care of from a class that extends PApplet.

• Do we have a class that extends PApplet?
Processing - drawing encapsulation

```java
public class Pong extends PApplet {
    Paddle paddle = new Paddle(500, 500, this);
}
```
Processing - drawing encapsulation

```java
import processing.core.*;

public class Pong extends PApplet {

    Paddle paddle = new Paddle(500, 500, this);

    public static void main(String[] args) {
        PApplet.main(new String[] { "Pong" });
    }

    public void settings(){
        size(500, 500);
    }

    public void setup(){
        background(0);
    }

    public void draw(){
        background(0);
        paddle.draw();
    }

    public void keyPressed(){
        if (keyCode == LEFT){
            paddle.moveLeft();
        } else if (keyCode == RIGHT){
            paddle.moveRight();
        }
    }
}
```
import processing.core.*;

public class Paddle {
    int gameWidth;
    int gameHeight;
    int width = 100;
    int height = 20;
    int xPos;
    int yPos;
    int speed = 10;

    PApplet canvas;

    Paddle(int gameWidth, int gameHeight, PApplet canvas){
        this.gameWidth = gameWidth;
        this.gameHeight = gameHeight;
        xPos = gameWidth / 2;
        yPos = gameHeight - height;
        this.canvas = canvas;
    }

    void moveLeft(){
        xPos -= speed;
        if (xPos < 0){
            xPos = 0;
        }
    }

    void moveRight(){
        xPos += speed;
        if (xPos + width > gameWidth){
            xPos = gameWidth - width;
        }
    }

    void draw(){
        canvas.rect(xPos, yPos, width, height);
    }
}