CS206

Exceptions, Scope & Restaurants
Exceptions

• Try-catch or throws
  • Catch must tell user what / why
• If throws, somewhere **in the program** it must be caught
• Try-catch should be as “tight” as possible
• Programs should never die on an exception
  • It is not acceptable to simply surround main with try-catch
public class Crash1b {
    private static void faulty() throws ArrayIndexOutOfBoundsException {
        int[] a = { 10, 20, 30, 40, 50 };  
        for (int i = 0; i <= 10; i++) {
            System.out.println(a[i]);
        }
    }

    public static void main(String[] args) throws ArrayIndexOutOfBoundsException {
        faulty();
        System.out.println("Done printing the array!");
    }
}
public class Crash1a
{
    static int[] a = { 10, 20, 30, 40, 50 };  
    public static void main(String[] args) {
        try {
            for (int i = 0; i <= 10; i++) {
                System.out.println(a[i]);
            }
        }
        System.out.println("Done printing the array!");
    }
}

Exceptions:
Good = Tight + Specific

```java
public class Crash1 {
    
    static int[] a = { 10, 20, 30, 40, 50 };
    public static void main(String[] args) {
        for (int i = 0; i <= 10; i++) {
            try {
                System.out.println(a[i]);
            } catch (ArrayIndexOutOfBoundsException aoe) {
                System.err.println(aoe.toString() + "Quitting");
                System.exit(0);
            }
        }
        System.out.println("Done printing the array!");
    }
}
```
Exceptions
OK — but not good or great

```java
import java.util.Scanner;

public class Crash2 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int prev=10;
        while (prev>=0) {
            System.out.print("Enter a number: ");
            String line = in.nextLine();
            try {
                int data = Integer.parseInt(line);
                System.out.println(data + " / " + prev + " = " + (data / prev));
                prev=data;
            } catch (NumberFormatException e) {
                System.out.println("That's not a number!");
            }
        }
        in.close();
    }
}
```

try could be “tighter” but it would be inconvenient
Multiple catch clauses

```java
import java.util.Scanner;
public class Crash2 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int prev=10;
        while (prev>=0) {
            System.out.print("Enter a number: ");
            String line = in.nextLine();
            try {
                int data = Integer.parseInt(line);
                System.out.println(data + " / " + prev + " = " + (data / prev));
                prev=data;
            } catch(NumberFormatException e) {
                System.out.println("That's not a number!");
            } catch (ArithmeticException ae) {
                System.err.println("MATH Problem " + ae.toString());
            }
        }
        in.close();
    }
}
```

THIS IS NOT A GOOD WAY TO HANDLE DIVISION BY ZERO
Scope

• Scope is used to define the "lifetime" of a variable.
• "Global" means variable always available from anywhere
  • public static
• "Local" means variable only available in a specific place.
  • { } delimit scope
• Each scope is aware of its variables
• Variables defined within a scope die at the end of the scope.
• Java does not allow var name re-use in enclosing scopes
public class Scoper {
    int var = 1;
    public void scopetest(int vv) {
        System.out.println(var);
        {
            int var = vv+2;
            System.out.println(var);
        }
        System.out.println(var);
    }
    public static void main(String args[]) {
        Scoper s = new Scoper();
        s.scopetest(2);
    }
}

Print?
GT Restaurant (simplified)

- GT offers 3 food types
  - drink: orangina, coffee, ...
  - main course: burger, hot dog, ...
    - NEW: Gluten Free: burger on lettuce
  - salad: spinach, cobb, ...
- At the start of each day, GT decides what will be offered that day and how much is available, price & cost
- Special Deal: “The Trio”. One each of drink, main & salad for the price of the two highest priced items
- During day:
  - Order either trio or one item
    - How many
    - if run out, remove from list
    - if order would use more than available, reject
- End of day print out:
  - leftover food
  - Cost, Revenue, Profit.
Considerations to implement GT

• How to store available menu items
  • GT wants to use only 1 data structure
• How to represent available menu items
• How to represent “The Trio”
• How/what to update on each order
public class GTRestaurant {

    public void addItem(Object o) {
    }

    public boolean doOrder(int id, int count) {
        return true;
    }

    public boolean doOrder(Trio aTrio) {
        return true;
    }

    public void endOfDay()
    {
        System.out.println("No cost, no revenue, no food");
    }

}
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        GTRestaurant gtr = new GTRestaurant();
        // Add items at start of day
        gtr.addItem(new Drink(1, "sip orangina", 1.50, 0.45, 20, 20));
        gtr.addItem(new Drink(2, "small coffee", 2.99, 0.20, 4, 250));
        gtr.addItem(new MainC(101, "hamburger", 3.50, 2.20, 30));
        gtr.addItem(new MainC(102, "hot dog", 2.45, 1.05, 40));
        gtr.addItem(new MainCGF(103, "burger on lett", 5.99, 0.40, 2000));
        gtr.addItem(new Salad(201, "spinach", 2.50, 1.99, "Vinegar", 20));
        gtr.addItem(new Salad(202, "cobb", 4.99, 1.99, "lemon juice", 200));
        gtr.doOrder(1, 2);
        gtr.doOrder(101, 5);
        gtr.doOrder(new Trio(1, 101, 201, 5));
        gtr.endOfDay();
    }
}
MenuItems
Class Hierarchy

Class or Interface?

Menu Item

Drink

Main Course

Main Course GF

Salad

Trio
Interfaces

• A class is what an object is
• An interface is what an object does
Interface

- A collection of method signatures with no bodies
- No modifier - implicitly public
- No instance variables except for constants (static final)
- No constructors
- Can not be instantiated
More on Interfaces

• An interface is not a class!
  □ cannot be instantiated
  □ incomplete specification

• An interface is a type
  □ a class that implements an interface is a subtype of the interface

• Classes may implement several interfaces

• Classes can only extend one class
public interface Shape {
    double area();
}

public class Circle implements Shape {
    private Point center; private double radius;
    //constructors and getters not shown
    public double area(){return Math.PI*radius*radius;}
}

public class Square implements Shape {
    private double sideLength;
    //constructors and getters not shown
    public double area(){return sideLength*sideLength;}
}
public interface Transportable {
    int volume();
    boolean isHazardous();
}

class Box implements Transportable {
    private int height, width, depth;
    // constructors and getters not shown
    public int volume(){
        return height*width*depth;
    } 
    public boolean isHazardous(){return false;}
}