1 Administrative Topics

• Return quizzes.

2 More about Inheritance and polymorphism

• Summary from Friday:
  – A class inherits all the methods of its ancestor classes, all the way up the Object class.
  – A subclass can override the inherited methods by defining a method with the same header in the subclass.

• Fields are also inherited by subclasses. However, if the fields are private in the superclass, then the subclass inherits them but can’t access them.

• Let’s try another line of code to see what happens:
  
  Object o = new Die();

  – What do you think will happen when we try to execute it? Error? [run it]
  – Since Die is a subclass of Object, every Die “is an” Object and so you can do such assignments. A variable of type A can be assigned any object of type A or of a type that is a subclass of A.
• How do you tell Java that you want your class to be a subclass of another class?
  – Use the `extends` keyword:
    ```java
    public class Die extends Object
    {
        ...
    }
    ```
  – But you don’t have to explicitly extend any class and in that case the Java compiler automatically puts in `extends Object` for you.

3 Static fields and methods

• Static methods belong to a class, not to an object. That is, they are things you ask classes to do, not objects.
  When `main` started, there was a class `Die` but there was not a `Die` object created yet. We asked the class `Die` to execute the main method.

• Where else have we seen static methods? [Math.random]

• Can have static fields as well that belong to the class. All objects of that class have access to the static variable; they don’t have their own copies of it like instance fields.
  Where have we seen static fields? [System.out]

4 Test for equality

• Note that comparing content and comparing references are different things.

• If we have two objects (such as Strings or Dice), and we compare them using `==`, then the result will only be True if both objects are the SAME object (meaning the same square block on the heap).

• If we want to compare the contents of two objects, then the standard approach is to implement an `equals` method that does the comparison.

• E.g. For the Die, we want to compare the up-facing sides of both dice:
```java
@Override
public boolean equals(Object other) {
    Die d = (Die) other;
    return this.sideUp == d.sideUp;
}
```

- The equals method exists on the Object class, and, to play nice with others, the proper course of action is to have the signature:

  ```java
  public boolean equals(Object other)
  ```

- which means this method really shouldn't crash if the other reference is null or if the it is an object, but it isn’t a Die. Java lets us check the type of each object, so we should take advantage of that and update our code:

  ```java
  @Override
  public boolean equals(Object other) {
      if (other == null) {
          return false;
      }
      if (this.getClass() != other.getClass()) {
          return false;
      }
      Die d = (Die) other;
      return this.sideUp == d.sideUp;
  }
  ```

## 5 Arrays

- Suppose you are playing 5 dice, as in the game of Yahtzee. Think about rolling all 5 dice and then looking for a 5-die straight.

- How do you keep track of the dice? You could have variables die1, die2, die3, die4, die5. Check the 5 dice values to see which of the value 1...6 appear. If all but 1 or 6 appear, you are done.

- Problem: Ugly code for rolling the dice and even uglier code for looking for a straight.

- Better: Use an ArrayList like you did in Lab 1.
• Disadvantage compared to Python list: notation is uglier. You can’t use the bracket notation.

• Java does have something like a Python list. It is called an array.

• An array is like an array list but:
  • It uses different notation
  • It has a fixed number of slots that you specify when you create the array.

• In Java, there are a variety of ways of creating a new array. One way is to first create the array and then fill it:
  ```java
  Die[] dice = new Die[5];
  for (int i = 0; i < 5; i++) {
      System.out.print(dice[i] + " ");
  }
  ```

• Notice that the array was created but the slots were not filled with Die objects. Instead, it is full of the keyword null. We need to create 5 new Die objects and add them to the array.
  ```java
  for (int i = 0; i < 5; i++) {
      dice[i] = new Die();
  }
  ```

• Note that you have to specify the array size when you create it. Java arrays are fixed size and cannot be increased or decreased!

• Arrays can have size up to 2 billion (more precisely, $2^{31} - 1$).

• To save space, the Java language creators added a second way to create an array, for when you know ahead of time all the values to be placed in the array initially:
  ```java
  int[] values = new int[] {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9};
  int[] values = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9};
  ```

• Let’s now use arrays to re-implement the quiz. [go over the code in Yahtzee.java]
```java
public static void main(String[] args) {
    Die[] dice = new Die[5];
    for (int i = 0; i < 5; i++)
        dice[i] = new Die();
    for (Die d : dice)
        d.roll();
    for (int i = 0; i < 5; i++)
        System.out.println(dice[i].getSideUp());
}
```

- What do you do if you created an array of size 10 and then, in the middle of the program, an 11th item needs to be added to the array.

6 2D Arrays

- What if you need bigger arrays than 2 billion? One way is to have arrays of arrays:

```java
int[][] matrix = new int[5][4];
```

This creates an array with 5 slots, each of which is an array of length 4 [draw a picture with pointers]

- Let’s play with this matrix. What do the following lines of code do?

```java
matrix[0][1] = 3;
for (int j = 0; j < 4; j++)
    matrix[0][j] = 1;
for (int i = 0; i < matrix.length; i++)
    for (int j = 0; j < matrix[0].length; j++)
        matrix[i][j] = i+j;
```

- We don’t need all inner arrays to be the same length:

```java
int[][] m = new int[2][];
m[0] = new int[] {1, 2, 3};
m[1] = new int[] {1, 2};
```

7 PSVM

- `public static void main`
By now you should understand pretty much everything in the header of the main method:

```java
public static void main(String[] args)
```

- What does that line tell you?
- It is a public method so that anyone can call it.
- It is static so you don’t need to create an object to call it.
- It has return type void so it doesn’t return anything.
- It takes an array of String objects as its parameter. That array has the name ”args”.
- Until now, we’ve ignored the args parameter, but you will need it for the project this week. It is used as a way of passing in values on the command line.
- Let’s look at a simple example:

```java
class CLTester {
    public static void main(String[] args) {
        System.out.println(args[0] + args[1]);
    }
}
```

- Now let’s execute this via ”java CLTester Hi There” [Do it]
- But what if you wanted to enter two numbers on the command line and then add them? Will the code work if I run ”java CLTester 3 4”? [Do it]

```java
class CLTester {
    public static void main(String[] args) {
        System.out.println(args[0] + args[1]);
    }
}
```

- The args are treated as strings, even if you run ”java CLTester 3 4”. To add them as integers, you need to convert the strings to integers. Here’s how you can do it:

```java
class CLTester {
    public static void main(String[] args) {
        int arg0 = Integer.parseInt(args[0]);
```
int arg1 = Integer.parseInt(args[1]);
System.out.println(arg0 + arg1);