Announcements

- If you were not here last lecture, rejoin the Tophat course at this join code: 828859
- Midterm 1 Date: 10/02
    - Soon I will release specific sections in Zybooks that I recommend you study
- Program 3 due date will be pushed back...

Midterm 1

Types of questions to expect:

- Multiple choice
  - Majority of the questions
  - Approximately 60% of your test grade
- Free response: read a block of code and figure out the output
  - Will have a few of these
  - Approximately 30% of your test grade
- Free response: write code to solve some problem
  - Will be syntactically forgiving but not too forgiving…
  - Only one question
  - Approximately 10% of your test grade

Midterm 1 (cont.)

Midterm Material:
- Memory
- Expected to know everything in Ch's 1 - 4
    - Will not test on this material explicitly, but you can certainly expect it to be part of questions that test other material
- Ch 5
    - Method terminology
    - Understand the different parts of a method signature
        - Access specifiers: public, private, protected, default(package)
        - static
        - return type: void, int, double, char etc..

Midterm 1 (cont.)

- Ch 5 (cont.)
    - how to use a static method vs use a non-static method
    - Method overloading
    - Reference Types and methods (pass by value)
- Ch 6
    - How to create an array
    - How to access array elements
    - Iterating through arrays
    - Why accessing elements in an array is efficient
    - Two Dimensional Arrays
        - how to create, access elements etc.

Midterm 1 (cont.)

- Ch 7
    - Object vs Class
    - How to create a class
    - How to create an object
    - Encapsulation and Abstraction
    - Mutators and Accessors
    - Private Helper methods
    - Constructors
    - Constructor overloading
    - primitive vs reference types
    - Reference Types and their relation to objects
    - 'this' implicit parameter

Midterm 1 (cont.)

- Ch 7 (cont.)
  - Wrapper Classes
  - ArrayList
  - Static Fields and Methods
  - Garbage Collection
  - Stack vs Heap (maybe)
- Ch 8
  - Derived Class relation to Base class
  - Overriding member methods
  - Object class

Inheritance

- **Derived class** (or **subclass**) refers to a class that is derived from another class that is known as a **base class** (or **superclass**)
- The derived class is said to inherit the properties of its base class, concept commonly called **inheritance**
- “Inherit the properties” means the derived class has access to all of the same public members as its corresponding base class

Let’s look at an example...

Inheritance (cont.)

- While a derived class may have access to the public members of the base class, it does NOT have access to the private members of the base class
- **Protected** is an access modifier that provides access to derived classes and other classes in the same package, but not by anyone else

Access Modifiers (revisited)

- **Private**: A variable or method that is private can only be accessed from within the class that it was declared (Most restrictive)
- **Package**: Can only be accessed by another class from within the same directory (package) as the class with this variable or method is declared
- **Protected**: Can be accessed by a derived class or by another class from within the same directory (package) as the class with this variable or method is declared
- **Public**: Can be accessed from any class (least restrictive)

Let’s apply this knowledge...

Inheritance (cont.)

- A derived class may provide its own definition for a base class method, this is known as: **Overriding**, it does this by using the annotation: **@Override**
- **Annotations** are optional notes beginning with the '@' symbol that can provide the compiler with useful information
    - In this case @Override lets the compiler know that a derived class will be re-defining a base-classes method
- The @Override annotation is considered best practice, but is technically not necessary, what it does is warn you at compile time about possible mistakes you are making while overriding the method

Let's look at an example..

Inheritance (cont.)

- Do not confuse “Override” with “Overload”, they are distinctly different
- Overloading allows for methods of the same name to exist, they simply must have different parameters types

Let’s look at an example...

Inheritance (cont.)

- You can still access an overridden method by using the keyword: **super**
- Super is used to access class members of an object's base class (superclass)

Inheritance (cont.)

You can see the hierarchy of inheritance in Oracle’s documentation:

Java SE 11 ArrayList

Inheritance (cont.)

- The **Object class** serves as the base class for all other classes and does not have a super class
- All classes therefore have a set of pre-implemented methods at their disposal
- Do not confuse the terms: “Object class” and “object” , they are different
- Recall that “object” refers to an instance of a class

Inheritance (cont.)

Common methods defined in the Object class include:

- **toString()** - returns a string representation of the object
- **equals(otherObject)** - Compares an Object to another object and returns true if both variables reference the same object

Let’s look at java’s Object class documentation for further methods...

Exceptions

- **Error-checking code** is code a programmer writes to detect and handle errors that occur during program execution.
- An **exception** is a circumstance that a program was not designed to handle, such as if the user enters a negative height
- The **try, throw and catch** keywords are known as **exception-handling constructs**

Exceptions (cont.)

Why use try, throw and catch?

- These exception-handling constructs are designed to keep error-checking code separate and to reduce redundant checks
- Can use if-else statements to accomplish the same functionality, except now it can get confusing between normal code and error-checking code.
- If-else statement are also subject to redundant checks if you are not careful
    - Having redundant checks is inefficient

Exceptions (cont.)

- A try block surrounds normal code, which is exited immediately if a throw statement executes
- A throw statement appears with a try block; if reached, execution jumps immediately to the end of the try block.
    - Code should be written so only error situations reach a throw statement
    - The throw statement must provide an object of type Throwable