Announcements

- If you were not here last lecture, rejoin the Tophat course at this join code: 828859
- Monday 9/30 will be review for your midterm, as long as you have questions… if you do not have questions then I will go into lecture
- Program 3 due date has been adjusted to 10/7 at Midnight (not right before class), it will be assigned soon

Quiz

Inheritance

```
class Foo {
        private int x = 5;
        protected int y = 6;
        public int z = 7;
}

 // The bar class now has access to the variables “y” and “z”, but not x...
class Bar extends Foo {

}
```

“extends” is the key word that allows a subclass to inherit from a superclass

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.)

Some notes on using the keyword “super”

- You do not need to call super to access the inherited members
  - But it is useful (when using an IDE) having all of the inherited members displayed in one place
- You can call the base class’s constructor by calling: super()
  - This call to the super class constructor must be done with the subclass constructor

Let’s look at an example...

Briefly moving away from inheritance...

We will come back to inheritance, but for now let’s move on to:

1. Exceptions
2. Streams

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 within 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** for example an object of type Exception
- A **catch** clause immediately follows a try block.
  - It is only reached if the an exception is thrown within the try block, this is known as “catching” the exception

Exceptions (cont.)

Example of syntax:

```
try {
        If (true) {
                Throw new Exception(“Something was invalid”);
        }
}
catch (Exception excpt) {
        System.out.println(excpt.getMessage());
}
```

Exceptions (cont.)

Throwing exceptions within methods:

- If a method throws an exception not handled with in the method, a programmer must include a **throws clause** with in the method declaration, by appending “**throws Exception**”
  - public void methodName() throws Exception {}

Exceptions (cont.)

```
class Foo {
    public void someMethod() throws Exception {
        if (true) {
            throw new Exception(“Something is invalid”);
        }
    }
}
class Main {
    public static void main(String[] args) {
        Foo bar = new Foo();
        try {
            bar.someMethod();
        }
        Catch (Exception excpt) {
            System.out.println(excpt.getMessage());
        }
    }
}
```

Exceptions (cont.)

- A **checked exception** is an exception that a programmer should be able to anticipate and appropriately handle.
  - An example of a checked exception is **Exception** class and several of its subclasses
- An **unchecked exception** is an exception that results from a hardware or logic error and is typically not anticipated or handled appropriately.
  - Such exceptions should terminate the program immediately

Exceptions (cont.)

- Unchecked exceptions are comprised of the **Error and RuntimeException** classes and their subclasses
- Examples of Unchecked exceptions include:
    - **NullPointerException** - Indicates a null reference
    - **IndexOutOfBoundsException** - Indicates that an index is outside the appropriate range
    - **ArithmeticException** - Indicates the occurrence of an exceptional arithmetic condition (i.e. divide by zero)
    - **IOError** - Indicates the failure of an I/O operation
    - **ClassCastException** - Indicates an invalid attempt to cast an object to type of which the object is not an instance (i.e. casting a Double to a String)
    - **IllegalArgumentException** - Thrown by a method to indicate an illegal or inappropriate argument

Exceptions (cont.)

Let’s look at Oracle’s documentation for Exceptions...

Exceptions (cont.)

- You can have multiple “catch” statements.

Example :

```
try {

}
catch (ExceptionType1 ex1){

}
catch (ExceptionType2 ex2) {

}
```

etc….

Exceptions (cont.)

- Error checking and handling exceptions are most useful when dealing with unknown input, generally from a user or a file.
- A file input/output stream requires exception handling to ensure invalid or interrupted file operation exceptions are appropriately caught and handled
- The **FileReader** class provides an input stream that allows a programmer to read characters from a file.
  - Most FileReader methods throw exceptions of type: **IOException** and its’ subclasses
  - One such subclass is: **FileNotFoundException** which inherits from IOException and is thrown when a file cannot be opened for reading

Exceptions (cont.)

Let’s look at an example of File I/O exception handling...