Notes

No Class Mon Feb 19

Midterm Weds March 6th

? and ! on command line

! and ? are just regular characters in Java programs, Eclipse, etc
But they are SPECIAL in terminal windows ("command shells")…

Class Review

Java syntax

Computer Science concepts

How to design and write programs

Three main categories of material in CS 112: syntax, concepts, writing programs

- A little more Java syntax every day
- Concepts e.g. "references and objects", encapsulation
- How to design programs is mostly YOUR PRACTICE. We will do examples in class.

Syntax review

8 basic variable types: double chanceOfRain = 0.85;

Type conversions: automatic or with cast: float rain = (float) chanceOfRain;

Boolean operators: `&&, ||, !, ==, !=, >=, <=, >, <`

```
boolean done = (counter >= maxValue);
boolean programCanExit = done && !foundErrors;
```

Loops, conditionals

Syntax Review

Constants:

-17

- How to make long? Short? Byte?

110.34

- How to make float?

What are these?

- 'v'
- "w"

What are Boolean constants?

V: char. W: string. Booleans: true, false
Boolean: true or false

Syntax Review

Classes:

```
class MyClass {
  // Class variables
  long classMember;

  // Constructors
  MyClass(float x) { classMember = (long) x; }

  // Methods
  long getValue() { return classMember; }
}
```

Object creation and use:

```
MyClass anObject = new MyClass();
System.out.println( anObject.getValue() );
```

USEFUL ONLINE TUTORIAL: https://testautomationu.applitools.com/java-programming-course/

Last time…

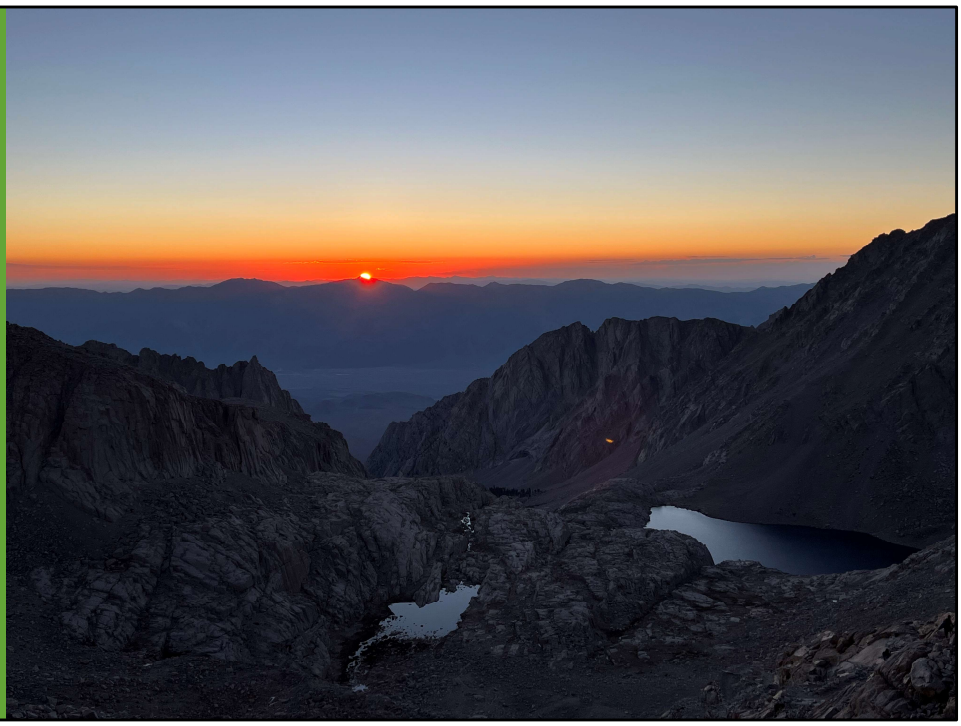
We discussed Objects vs References

And in the lab we worked with arrays, like

- `String[] args;`
- `Card[] allMyCards = new Card[52];`
- `allMyCards[0], …, allMyCards[51]`

Arrays are like in Python
Up to element #51, not #52

Method Overloading again



Method Overloading

Ok

```
class Student {
      void set(String name);
      void set(int idNumber);
      void set(Boolean hasGraduated);
```

How about?

```
      int get(); // return idNumber
      String get(); // return name
      boolean get(); // return graduated
```

ONLY CAN OVERLOAD BY ARGUMENT, NOT RETURN TYPE
Too much automatic conversion if based on return types:
boolean status(); long status(); Sys.out.println(status());

Method Overloading

How about this?

```
class Difficult {
     void fcn(int a, double b) { System.out.println("int, dbl"); }
     void fcn(double a, int b) { System.out.println("dbl, int"); }
}
…
     Difficult dd = new Difficult();
     dd.fcn(2, 3);
```

Not allowed! "Ambiguous method"

One more thing!

Can use a function call anywhere that its return type is needed

```
class Company {
  int status() { return statusVariable; }
  void printInt(int value) { System.out… }

  void monthlyReport(…) {
       printInt( status() );
  }
}
```

Variable Scope

Scope of a variable

"Scope" of a variable is where in your source code the variable can be used.

When a variable is defined in a class method, it can be used anywhere inside the method (below where it is defined)

- But not outside the method

```
class myClass {
 void myFunction() {
  int x = 2;
  System.out.println("value is " + x);
 }
 void otherFunction() { System.out.println(x); }
}
```

GREEN is ok
RED is ERROR

Scope of a variable

When a variable is defined in a "code block", it can be used anywhere inside the code block.

```
class myClass {
 void myFunction() {
  if (time == 0) {
    int x = 2;
    ...
  }
  System.out.println("value is " + x);
 }
```

ERROR

Scope of a variable

When a variable is defined at the "class level" ("class variable", "instance variable"), it can be used:

- anywhere inside the class, directly (in any member function)
- in other classes, accessed through an object of the class (if "accessible")

ACCESSIBLE = public (or in same package and not private)

Scope of a variable

```
class A {
     public int memberOfA = 5;
     private double privateMember = memberOfA;

     String toString() { return new String(memberOfA); }
}

class B {
 int bVal;

  B() {
     A myAObj = new A();
     bVal = 10 + myAObj.memberOfA;
   }
  int problem() { return 20 + myAObj.memberOfA; }
}
```

`this` keyword

`this` lets source code refer to the current object.

We can access a "class level" instance variable even if hidden by a "method variable".

```
class myClass {
 int x = 1;
 void myFunction() {
    System.out.println("x is " + x);
    int x = 2;
    System.out.println("x is " + x);
    System.out.println("this.x is " + this.x);
 }
```

Other use of THIS is for one constructor to call another
PRINTS 1 <cr> 2 <cr> 1

Scope of a variable

- How about this?

```
class myClass {
 void myFunction() {
  int x = 1;
  if (time == 0) {
    int x = 2;
  }
  System.out.println("x is " + x);
 }
}
```

Not legal Java. (Yes legal C++)

Static variables and methods

Remember…

Two kinds of variables in Java

- Class variables (= instance variables)
- Method variables (= local variables)

A class variable can be used in *any* of a class's methods

Static Instance Variables and Methods

A `static` variable is an instance variable that "belongs to the class", not to any particular object. Every object shares the same instance of the variable.

```
Math.PI
```

A static variable can be accessed through its class name, or an object name.

Determining if a variable should be static is an important design decision

- Do values vary for different objects in the class, or not?
- Changing value of a static variable changes it for all objects in the class!

static methods only can operate on static (and locally declared) variables, not nonstatic class variables!

```
Math.cos()
```

Preferred to access thru class name!

Static Methods

```
public class Helper
{
   public static int cube(int num)
   {
      return num * num * num;
   }
}
```

Because it is declared as static, the `cube` method can be invoked through the class name:

```
value = Helper.cube(4);
```


For good style, static methods SHOULD be called via the Class name, not an object name.

Static Class Members

The order of the modifiers can be interchanged, but by convention visibility modifiers come first: "`public static`"

Recall that the `main()` method is static – it is invoked by the Java interpreter without creating an object

This is why `main()` cannot access ordinary class variables!

```
//********************************************************************
//  Slogan.java       Author: Lewis/Loftus
//
//  Represents a single slogan string.
//********************************************************************

public class Slogan
{
   private String phrase;
   private static int count = 0;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the slogan and counts the number of
   //  instances created.
   //-----------------------------------------------------------------
   public Slogan(String str)
   {
      phrase = str;
      count++;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns this slogan as a string.
   //-----------------------------------------------------------------
   public String toString()
   {
      return phrase;
   }

   //-----------------------------------------------------------------
   //  Returns the number of instances of this class that have been
   //  created.
   //-----------------------------------------------------------------
   public static int getCount()
   {
      return count;
   }
}
```

```
//********************************************************************
//  SloganCounter.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the static modifier.
//********************************************************************

public class SloganCounter
{
   //-----------------------------------------------------------------
   //  Creates several Slogan objects and prints the number of
   //  objects that were created.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Slogan obj;

      obj = new Slogan("Remember the Alamo.");
      System.out.println(obj);

      obj = new Slogan("Don't Worry. Be Happy.");
      System.out.println(obj);
```

continue

continue

```
        obj = new Slo
        System.out.pr

        obj = new Slo
        System.out.pr

        obj = new Slo                                          );
        System.out.pr

        System.out.println();
        System.out.println("Slogans created: " + Slogan.getCount());
    }
}
```

Output

```
Remember the Alamo.
Don't Worry. Be Happy.
Live Free or Die.
Talk is Cheap.
Write Once, Run Anywhere.

Slogans created: 5
```

Scope of a static variable?

Created before main() called
Exists until program terminates
Lifetime is the ENTIRE PROGRAM

Lab07