administrivia...
-Lab 0 posted
  -getting started with Eclipse
  -Java refresher
    -this will not count towards your grade

-TA office hours today, 12:15-5pm

-help sessions this Friday
  -9:40am | 10:45am | 11:50am
  -MEB 3225
last time...
we refer to unspecified integer quantities as $N$
- $N$ is the problem size
  - sorting an array of $N$ numbers
  - searching for an item in a set of $N$ items
  - inserting an item into a set of $N$ items

amount of work done for these operations usually depends on $N$
- work required is a function of $N$
As $N$ becomes large, complexity matters!

Choose an Algorithm

As $N$ becomes large, complexity matters!
phases of software development

- requirements gathering
  - read and understand assignment specs, ask questions

- planning | design | analysis
  - outline how to solve a problem, determine algorithms, write pseudocode

- construction
  - write code, debug **syntactic** errors

- testing
  - test thoroughly to find **semantic** errors and boundary cases

- maintenance
testing

-white-box
- test with knowledge of the program’s inner-workings
  — from the programmer’s perspective
  - *unit testing*, *boundary analysis*

-black-box
- test only with knowledge of the program’s interface
  — from the user’s perspective
  - *stress testing*

-test-first model
- write acceptance tests before writing any code
good coding style

-benefits the programmer and all other readers of the program

-components:
  -descriptive names (variables, methods, classes)
  -clear expressions, straightforward control flow
  -consistency, conventions, and language idioms
  -comments!

-well-written code is often smaller, has fewer errors, and is easier to extend and modify
today...
disclaimer: this class is *not* about teaching you Java
-variables

-control flow

-reference types

-misc.
variables
a variable is a piece of data in memory with:
- an identifier (name)
- a type

what is a type?
- a basic building block in a programming language
- determines what kind of data a variable holds, and what operations can be performed on it

Java defines eight primitive types
- byte, short, int, long, float, double, char, boolean
- each primitive type can hold a single value
  - 'r', 12, 2.64, true
declaration & initialization

- **declaring** a variable is stating that it exists
  - assigns the variable a type and name
    ```java
    boolean areWeThereYet;
    ```

- **initializing** a variable gives it an initial value, and is often combined with declaring
  ```java
  boolean areWeThereYet = false;
  ```

- variables declared as **final** are constant and cannot be changed after initialization
  ```java
  final int theMeaningOfLife = 42;
  ```
assignment

- after a variable has been declared we can assign it a new value with =

  areWeThereYet = true;

- we can use arithmetic expressions with an assignment

  age = currentYear - birthYear;
arithmetic operations

- explicitly supported on primitive types
  - binary operators
    - +, -, *, /, %
  - unary operators
    - (negation), ++ (increment), -- (decrement)

Java follows common order-of-operation rules

  unary ops : highest
  *, /, % : high
  +, - : low
  = : lowest
type conversion

-widening conversions convert data to another type that has the same or more bits of storage

  short  ->  int
  int     ->  long
  int     ->  float

-narrowing conversions convert data to another type that has fewer bits of storage, possibly losing information

  double  ->  float
  float   ->  int
type conversion

- Java uses widening conversion when an operator is applied to operands of different types (called promotion)

\[
\begin{align*}
2.2 \times 2 & \quad \text{evaluates to 4.4} \\
1.0 \div 2 & \quad \text{evaluates to 0.5} \\
\text{double } x = 2; & \quad \text{assigns 2.0 to } x \\
\text{“count = “ + 4} & \quad \text{evaluates to “count = 4”}
\end{align*}
\]
mixing types

- conversions are done on one operator at a time in the order the operators are evaluated

\[
\begin{align*}
3 / 2 & \times 3.0 + 8 / 3 & = 5.0 \\
2.0 & \times 4 / 5 + 6 / 4.0 & = 3.1
\end{align*}
\]
mixing types

- String concatenation has the same precedence as +/- and is evaluated left to right

\[
\begin{align*}
1 + "x" + 4 & \Rightarrow "1x4" \\
"2+3=" + 2 + 3 & \Rightarrow "2+3=23" \\
1 + 2 + "3" & \Rightarrow "33" \\
"2*3=" + 2 * 3 & \Rightarrow "2*3=6" \\
4 - 1 + "x" & \Rightarrow "3x" \\
"x" + 4 - 1 & \Rightarrow \text{error}
\end{align*}
\]
type casting

type casting tells Java to convert one type to another

-uses:

- convert an int to a double to force floating-point division
  double average = (double) 12 / 5;

- truncate a double to an int
  int feet = (int) (28.3 / 12.0);
assignment operators

- basic assignment operator
  =

- combined assignment/arithmetic operators
  +=, -=, *=, /=

- increment/decrement operators can be stand-alone statements

  int i = 3;
  i++;
  i--;
  ++i;
  --i;

  int i = 3;
  int j = i++;
  System.out.println(i+" " +j);

  int i = 3;
  int j = ++i;
  System.out.println(i+" " +j);
relational and logical ops

- results are always `boolean`

- relational ops supported for number and character types (and equality for `boolean`)
  - `>`, `<`, `>=`, `<=`, `==`, `!=`

- logical ops supported for `boolean`
  - `&&`, `||`, `!`

- precedence (all lower than arithmetic):
  - `>`, `<`, `>=`, `<=` : highest
  - `==`, `!=` : high
  - `&&` : low
  - `||` : lowest
control flow
-**control flow** determines how programs make decisions about what to do, and how many times to do it

- decision making: *if-else, switch-case*
- looping: *for, while, do-while*
- jumping: *break, continue, return*
- exceptions: *try-catch, throw*
switch statements

-similar to an if-else-if statement

switch(integer expression)
{
    case <integer literal>:
        list of statements...

    case <integer literal>:
        ...

}
**switch statements**

- execution begins on the ***case*** that matches the value of the switch variable

- execution continues until ______ is reached
  - even continues through other cases!
  - usually want a `break` after every case

- switches can use the `default` keyword
  - if no cases were hit, execute the `default` case
  - similar to an `else` at the end of a long line of `if-else-if`
exceptions

- an exception is a special event that interrupts the control of the program

- exceptions are “thrown” explicitly by the code

- use a try block to wrap any code that might throw an exception

- a catch block immediately follows a try block

- execution of the program jumps inside the catch block if an exception occurred within the try block
try
{
    FileReader in = new FileReader("fakefile.txt");
}
catch(FileNotFoundException e)
{
    System.out.println("file does not exist");
}
catch(Exception e) // a less specific error occurred
{
    System.err.println(e.getMessage());
}
throwing exceptions

if(arraySize < 0)
    throw new NegativeArraySizeException();
arr = new int[arraySize];

-why don’t we need an else?

-there are many many subclasses of exceptions…

-you can even define your own!

public class BadnessOccurred extends Exception
{   ...  }
reference types
-all non-primitive types are reference types

-a reference is a variable that stores the memory address where an object (a group of values) resides

```java
Point p1, p2, p3;
p1 = new Point(7,19);
p2 = p1;
```
reference declaration

- declaration of a reference variable only provides a name to reference an object — *it does not create an object*

-after `Point p1;` the value stored in `p1` is: ____

-the `new` keyword is used to construct an object

```java
Point p1 = new Point();
Point p2;
p2 = new Point();
```

-why are () needed?
operations on reference types

-operations on references: =, ==, !=
-equality operators compare addresses

-what does \texttt{p2 \texttt{==} p1} return?

Point \texttt{p1, p2, p3};
\texttt{p1} = \texttt{new Point(7,19)};
\texttt{p2} = \texttt{p1};
operations on reference types

-operations on objects: ., instanceof
  -the . operator is used to select a method that is applied to an object, or an individual component of an object

Point p1, p2, p3;
p1 = new Point(7,19);
p2 = p1;

what does p3.firstValue() return?
what does p1 instanceof Point return?
String

- **String** is the only reference type for which operator overloading is allowed (+ and +=)

- **String** objects are **immutable**

- To compare **String** objects use **equals** and **compareTo** methods — not ==, !=, <, or >
  - Why?

- Other useful **String** methods:
  - length, charAt, substring
arrays

-an array is a mechanism for storing a collection of identically typed entities

-in Java, arrays behave like objects

-the [] operator indexes an array, accessing an individual entity — bounds checking is performed automatically

-by default, array elements are initialized 0 (primitive types) and null (reference types)

```java
Point[] refArray = new Point[10];
double[] primArray = {3.14, 2.2, -9.8};
```
- the `ArrayList` class (from the Collections library) mimics an array and allows for dynamic expansion

- the `get`, `set` methods are used in place of `[]` for indexing

- the `add` method increases the size by one and adds a new item

- `ArrayList` may only be used with reference types

```java
ArrayList<String> a = new ArrayList<String>(1);
a.set(0, "hi");
a.add("there");
```
misc.
parameter passing

- Java uses **call-by-value** parameter passing
  - ie. a copy is created

- what does this mean for reference types?

```java
int i = 4;
modifyInt(i);
System.out.println(i); // prints 4
```

```java
Point p = new Point(1, 2);
modifyPoint(p);
System.out.println(p.x); // prints ????
```
**main**

-when a program is run, the **main** method is invoked

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

-the parameters of **main** can be set using **command-line arguments**

- more on this later!
classes & constructors

- A **class** consists of **fields** (aka. variables) that store data and **methods** that operate on that data.

- Fields and methods may be **public** or **private**.

- The **constructor** controls how an object is created and initialized.
  - Multiple constructors may be defined, taking different parameters.
  - If none is defined, a default constructor is generated:
    - Initializes primitive fields to 0, and reference fields to null.

---

**The difference between field and variable:**

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html
this

-this is a reference to the current object
-useful in avoiding self-assignment

Account account1;
Account account2;
...
account2 = account1;
account1.finalTransfer( account2 );

// transfer all money from rhs to current account
public void finalTransfer( Account rhs )
{
    dollars += rhs.dollars;
    rhs.dollars = 0;
}
this

-this is a reference to the current object
-useful in avoiding self-assignment

Account account1;
Account account2;
...
account2 = account1;
account1.finalTransfer( account2 );

// transfer all money from rhs to current account
public void finalTransfer( Account rhs )
{
    if ( this == rhs )
        return;
    dollars += rhs.dollars;
    rhs.dollars = 0;
}
next time...
- reading
  - chapters 3 & 4

- homework
  - assignment 1 up by 5pm
  - due next Thursday at 5pm
  - must complete on your own!

- no lab
  - happy MLK day!

- clicker questions start next Thursday