Objectives

- Understand when to use constants
- Understand difference between classes and objects
- Understand difference between objects and primitive data types

Recap: Data Type Sizes

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

fixed size, so finite capacity

Recap: Declaration and Assignment

Variable declaration is instruction to compiler
- reserve block of main memory large enough to store data type specified in declaration
- Variable name is specified by identifier
- Syntax:
  - typeName variableName;
  - typeName variableName = value;
  - can declare and assign in one step

Java first computes value on right side
Then assigns value to variable given on left side
x = 4 + 7;     // what's in x?

Recap: Assignment Statements

Here’s an occasional point of confusion:

```
a = 7;        // what's in a?
b = a;        // what's in b?
             // what's in a now???
System.out.println("a is " + a + "b is " +b);
a = 8;
System.out.println("a is " + a + "b is " +b);
```

Draw and fill in boxes for your variables at each time step if you’re confused
Recap: Expressions

- expression is combination of
  - one or more operators and operands
  - operator examples: +, *, /, ...
  - operand examples: numbers, variables, ...
- precedence: multiply/divide higher than add/subtract

Recap: Primitive Data Types: Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approx -3.4E38 7 sig. digits</td>
<td>approx 3.4E38 7 sig. digits</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 15 sig. digits</td>
<td>approx 1.7E308 15 sig. digits</td>
</tr>
</tbody>
</table>

- Primary primitives are **int** and **double**
  - three other integer types
  - one other real type

Recap: Converting Between Types

- Doubles can simply be assigned ints
  - `double socks = 1;`
  - ints are subset of doubles
- Casting: convert from one type to another with information loss
  - Converting from real to integer
    - `int shoes = (int) 1.5;`
  - Truncation: fractional part thrown away
    - `int shoes = (int) 1.75;`
  - Rounding: must be done explicitly
    - `shoes = Math.round(1.99);`

Recap: Primitive Data Types: Non-numeric

- Character type
  - named char
  - Java uses the Unicode character set so each char occupies 2 bytes of memory.
- Boolean type
  - named boolean
  - variables of type boolean have only two valid values
    - true and false
  - often represents whether particular condition is true
  - more generally represents any data that has two states
    - yes/no, on/off

What Changes, What Doesn’t?

```java
// Vroom.java Author: Tamara
// Playing with constants
public class Vroom {
    public static void main (String[] args) {
        double lightYears, milesAway;
        lightYears = 4.35; // to Alpha Centauri
        milesAway = lightYears * 186000 * 60 * 60 * 24 * 365;
        System.out.println("lightYears: " + lightYears + " milesAway: " + milesAway);
        lightYears = 68; // to Aldebaran
        milesAway = lightYears * 186000 * 60 * 60 * 24 * 365;
        System.out.println("lightYears: " + lightYears + " milesAway: " + milesAway);
    }
}
```

Constants

- Things that do not vary
  - unlike variables
  - will never change
- Syntax:
  - `final typeName variableName;`
  - `final typeName variableName = value;`
- Constant names in all upper case
  - Java convention, not compiler/syntax requirement
## Programming With Constants

```java
public static void main (String[] args)
{
    double lightYears, milesAway;
    final int LIGHTSPEED = 186000;
    final int SECONDS_PER_YEAR = 60*60*24*365;

    lightYears = 4.35; // to Alpha Centauri
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: "+ lightYears + "miles " + milesAway);

    lightYears = 68; // to Aldebaran
    milesAway = lightYears * LIGHTSPEED * SECONDS_PER_YEAR;
    System.out.println("lightYears: "+ lightYears + "miles " + milesAway);
}
```

## Avoiding Magic Numbers

- **magic numbers**: numeric constants directly in code
  - almost always bad idea!
    - hard to understand code
    - hard to make changes
    - typos possible
  - use constants instead

## Programming

- Programming is all about specifying
  - data that is to be manipulated or acted upon
  - operations that can act upon data
  - order in which operations are applied to data

- So far: specify data using primitive data types
  - come with pre-defined operations like +, -, *, and /

## Programming with Classes

- What if data we want to work with is more complex these few primitive data types?
  - We can make our own data type: create a class
    - specifies nature of data we want to work with
    - operations that can be performed on that kind of data
  - Operations defined within a class called methods
Programming with Classes

- Can have multiple variables of primitive types (int, double)
  - each has different name
  - each can have a different value
    ```java
    int x = 5;
    int y = 17;
    ```
- Similar for classes: can have multiple instances of class String
  - each has different name
  - each can have different value
    ```java
    String name = "Tamara Munzner";
    String computerName = "pangolin";
    ```

Programming with Objects

- Object: specific instance of a class
- Classes are templates for objects
  - programmers define classes
  - objects created from classes

Object Example

```java
public class StringTest {
    public static void main(String[] args) {
        String firstname;
        String lastname;
        firstname = new String("Kermit");
        lastname = new String("theFrog");
        System.out.println("I am not " + firstname + " " + lastname);
    }
}
```

- Variable declaration does not create objects!
  - just tells compiler to set aside spaces in memory with these names
  - Spaces will not actually hold the whole objects
    - will hold references: pointers to or addresses of objects
    - objects themselves will be somewhere else in memory
So firstname and lastname will not contain String objects
• contain references to String objects

Object Example
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = new String ("Kermit");
        lastname = new String ("theFrog");
        System.out.println("I am not " + firstname + " " + lastname);
    }
}

■ constructors

■ Constructor: method with same name as class
• always used with new
• actually creates object
• typically initializes with data

firstname = new String ("Kermit");

Object Example
public class StringTest
{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = new String ("Kermit");
        lastname = new String ("theFrog");
        System.out.println("I am not " + firstname + " " + lastname);
    }
}

■ new operator and String constructor method
■ instantiate (create) new instance of String class (a new String object)

Object Example
Now create new instance of the String class
• String object with data “Kermit”
• Puts object somewhere in memory
   • puts address of the object’s location in firstname:
     • firstname holds reference to String object with data “Kermit”

Object Example
firstname

New operator and String constructor method

expression on right side of assignment operator

String object
"Kermit"
Object Example

```java
public class StringTest {
    public static void main (String[] args) {
        String firstname;
        String lastname;
        System.out.println("I am not " + firstname + " " + lastname);
    }
}
```

- Can consolidate declaration, assignment
  - just like with primitive data types

Objects vs. Primitives

- references vs. direct storage

Class Libraries

- Before making new class yourself, check to see if someone else did it already
  - libraries written by other programmers
  - many built into Java
- Example
  - Java has single-character primitive data type
  - what if want to work with sequence of characters
  - String class already exists
API Documentation

- Online Java library documentation at http://java.sun.com/j2se/1.5.0/docs/api/
- textbook alone is only part of the story
- let's take a look!

- Everything we need to know: critical details
  - and often many things far beyond current need

- Classes in libraries are often referred to as Application Programming Interfaces
  - or just API

Some Available String Methods

- public String toUpperCase();
  - Returns a new string object identical to this object but with all the characters converted to upper case.

- public int length();
  - Returns the number of characters in this String object.

- public boolean equals(String otherString);
  - Returns true if this String object is the same as otherString and false otherwise.

- public char charAt(int index);
  - Returns the character at the given index. Note that the first character in the string is at index 0.

More String Methods

- public String replace(char oldChar, char newChar);
  - Returns a new String object where all instances of oldChar have been changed into newChar.

- public String substring(int beginIndex);
  - Returns new String object starting from beginIndex position

- public String substring(int beginIndex, int endIndex);
  - Returns new String object starting from beginIndex position and ending at endIndex position
  - up to but not including endIndex char:
    - substring(4, 7)  "o K"

<table>
<thead>
<tr>
<th>H e l l o K e r m i t F r o g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
</tbody>
</table>

Questions?