

Programming in Java

CSCI 2220

Java Basics

Data Types

- Java has two main categories of data types:
 - Primitive data types
 - Built in data types
 - Many very similar to C++ (int, double, char, etc.)
 - Variables holding primitive data types always hold the actual value, never a reference
 - Reference data types
 - Arrays and user defined data types (i.e. Classes, Interfaces)
 - Can only be accessed through reference variables

Primitive Data Types

- Integer data types
 - `byte` – 8 bits (values from -128 to +127)
 - `short` – 16 bits (-32768 to +32767)
 - `int` – 32 bits (very big numbers)
 - `long` – 64 bits (even bigger numbers)
- Characters
 - `char` – 16 bits, represented in unicode, not ASCII!
- Floating point data types
 - `float` – 4 bytes (-3.4×10^{38} to $+3.4 \times 10^{38}$)
 - `double` – 8 bytes (-1.7×10^{308} to 1.7×10^{308})
- Boolean data
 - `boolean`
 - can only have the value true or false
 - Unlike C++, cannot be cast to an int (or any other data type)

Operators

- Arithmetic
 - +, -, *, /, %, ++, --
- Logic
 - &, |, ^, ~, <<, >>, >>>
- Assignment
 - =, +=, -=, etc..
- Comparison
 - <, <=, >, >=, ==, !=
- Work just like in C++, except for special String support

Comparison Operators

- Note about comparison operators
 - Work just as you would expect for primitive data types
 - We will see that they do not always operate as you would think for reference data types

Control Structures

- **if/else, for, while, do/while, switch**
- Basically work the same as in C/C++

```
i = 0;
while(i < 10) {
    a += i;
    i++;
}
```

```
i = 0;
do {
    a += i;
    i++;
} while(i < 10);
```

```
for(i = 0; i < 10; i++) {
    a += i;
}
```

```
if(a > 3) {
    a = 3;
}
else {
    a = 0;
}
```

```
switch(i) {
    case 1:
        string = "foo";
    case 2:
        string = "bar";
    default:
        string = "";
}
```

Control Structures

- Java also has support for **continue** and **break** keywords
- Again, work very similar to C/C++

```
for(i = 0; i < 10; i++) {  
    a += i;  
    if(a > 100)  
        break;  
}
```

```
for(i = 0; i < 10; i++) {  
    if(i == 5)  
        continue;  
    a += i;  
}
```

- Also note: **switch** statements require the condition variable to be a **char**, **byte**, **short** or **int**

Reference Data Types

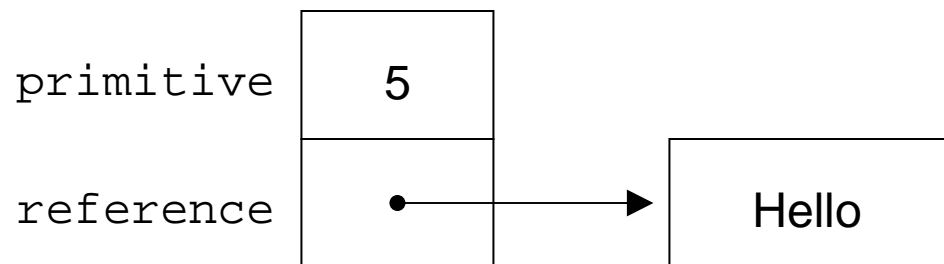
- Key difference between primitive and reference data types is how they are represented
- Primitive type variables hold the actual value of the variable
- Reference type variables hold the value of a reference to the object

Example

- Variable declarations:

```
int primitive = 5;  
String reference = "Hello";
```

- Memory representation:



Arrays

- In Java, arrays are reference data types
- You can define an array of any data type (primitive or reference type)
- Special support built into the language for accessing information such as the length of an array
- Automatic bound checking at run time

Declaring Array Variables

- Declare array variables:

```
int myNumbers[];  
String myStrings[];
```

- This just creates the references
- You must explicitly create the array object in order to use it

Creating Array Objects

- Create array objects:

```
myNumbers = new int[10];  
myStrings = new String[10];
```

- In the creation of any reference data object, the **new** operator is almost always used
 - String objects can be created from String literals as we've seen
 - Array objects can also be created using Array literals

```
myNumbers = {1, 2, 3, 4, 5};
```

- Note: in the first example, myStrings is a reference to an array of references

Accessing Array Elements

- Just like in C/C++

```
myNumbers[0] = 5;  
myStrings[4] = "foo";
```

- Arrays also have a special length field which can be accessed to determine the size of an array

```
for(int i = 0; i < myNumbers.length; i++)  
    myNumbers[i] = i;
```

Wrapper Classes

- Each primitive data type has a corresponding “Wrapper Class” reference data type
- Can be used to represent primitive data values as reference objects when it is necessary to do so
- **Byte, Short, Integer, Long, Float, Double, Boolean, Character**

What are classes?

- User defined data types
- Classes can contain
 - Fields: variables that store information about the object (can be primitive or reference variables)
 - Methods: functions or operations that you can perform on a data object

Example

```
public class Circle {
    static final double PI = 3.14;
    double radius;

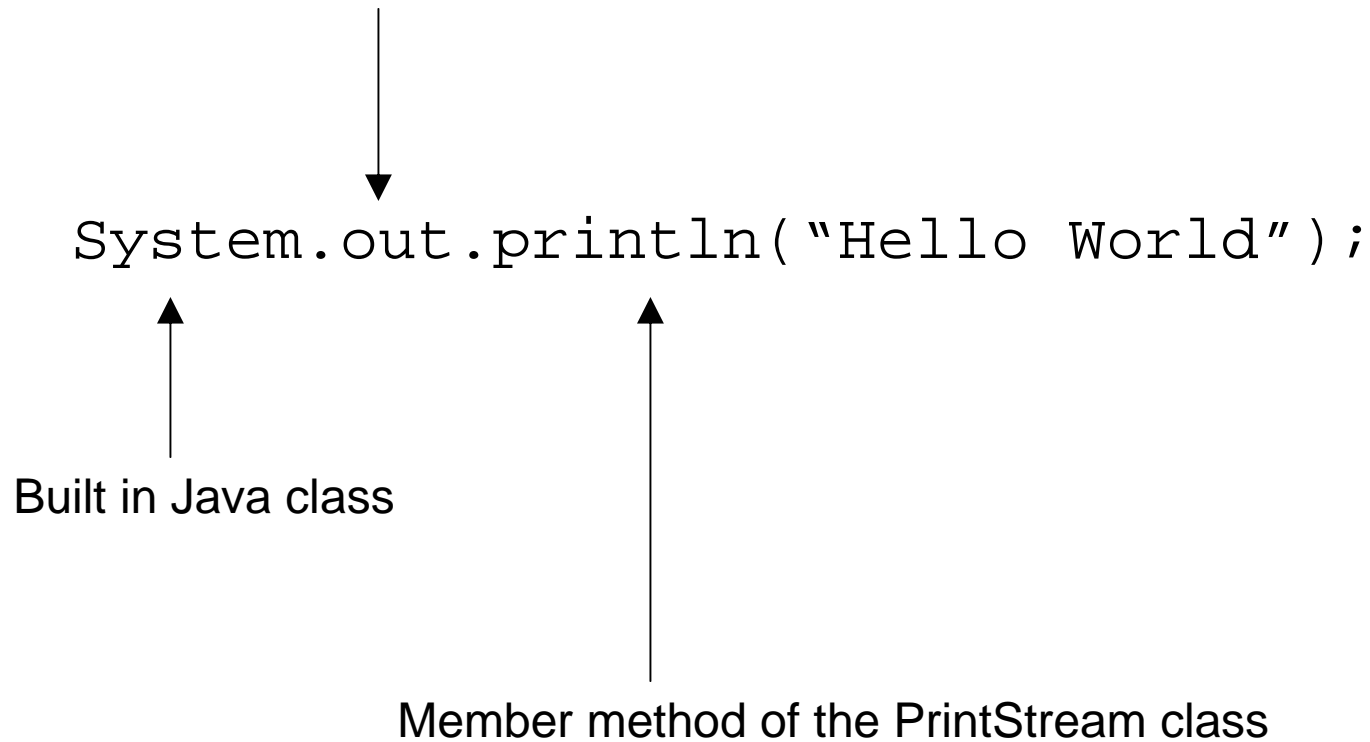
    public double area()
    {
        . . .
    }

    public double circumference()
    {
        . . .
    }
}
```

- Each instance of a Circle object contains the fields and methods shown
- **static**
 - The static keyword signifies that the class contains only one copy of a given member variable
 - Non static member variables have their own copy per instance of the class

Example

Static member variable (field) of the System class:
PrintStream object that points to standard out



Back to our Wrappers

- Each wrapper class contains a number of methods and also static methods that are potentially useful
- Examples:
 - `Character.toLowerCase(ch)`
 - `Character.isLetter(ch)`
- See the Java API for more details:
 - <http://java.sun.com/j2se/1.4.2/docs/api/index.html>

Strings

- In Java, Strings are reference data types
- They are among many built-in classes in the Java language (such as the wrapper classes)
 - However, they do not work exactly like all classes
 - Additional support is built in for them such as String operators and String literals

Creating Strings

- As mentioned before, creating reference objects in Java requires the use of the **new** operator
- Strings can be created this way:

```
String myString = new String("foo");
```

- However, String literals can also be used

```
String myString = "foo";
```

String Operators

- The + operator contains special support for String data types:

```
myString = "foo" + "bar";
```

- And also the += operator:

```
myString = "foo";  
myString += "bar";
```

Comparing Strings

- As mentioned before, the `==` operator does not always work as expected with reference data types

- Example:

```
String string1 = "foo";  
String string2 = string1;  
if(string1 == string2)  
    System.out.println("Yes");
```

- Evaluates to true iff `string1` and `string2` both contain a reference to the same memory location

Solution

- The String class contains built in methods that will compare the two strings rather than the two references:

```
String string1 = "foo";  
String string2 = string1;  
if(string1.equals(string2))  
    System.out.println("Yes");
```

```
String string1 = "foo";  
String string2 = string1;  
if(string1.equalsIgnoreCase(string2))  
    System.out.println("Yes");
```

More String comparison

```
string1.compareTo(string2);
```

- Returns a negative integer when string1 is “alphabetically” before string2
- Note: “alphabetically” means we are considering the unicode value of a character
- The character with the smaller unicode value is considered to come first

Accessing String characters

```
string1.charAt(0);
```

- Returns a Character reference object (not a char primitive variable)
- Throws a `StringIndexOutOfBoundsException` if given an index value that is not within the range of the length of the string
 - We will see more on exceptions later
 - For now you do not need to worry about this

Other useful String methods

```
string1.length();  
string1.indexOf('a');  
string1.substring(5);  
string1.substring(5,8);  
string1.toCharArray();  
string1.toUpperCase();  
.  
.  
.
```

- See more at the Java API site
 - <http://java.sun.com/j2se/1.4.2/docs/api/index.html>

StringTokenizer

```
String text = "To be or not to be";
StringTokenizer st = new StringTokenizer();
While(st.hasMoreTokens()) {
    System.out.println("The next word is " +
        st.nextToken());
}
```

- Can be used to split strings into “tokens” using a fixed delimiter
- Default delimiter is a space
- Can also specify custom delimiters
- Must import `java.util.StringTokenizer`

Homework 1

- Homework 1 is posted on the course website
- Due Feb. 2 at 11:55 PM
- Submit via WebCT
- WebCT discussion boards are up