Java Primer II

CMSC 202

Expressions

- An ***expression*** is a construct made up of variables, operators, and method invocations, that evaluates to a single value.
- For example:

```
int cadence = 0;

anArray[0] = 100;

System.out.println("Element 1 at index 0: " + anArray[0]);

int result = 1 + 2;

System.out.println(x == y ? "equal" :"not equal");
```

Statements

- ***Statements*** are roughly equivalent to sentences in natural languages. A ***statement*** forms a complete unit of execution.
- Two types of statements:
  - Expression statements – end with a semicolon ';'
    - Assignment expressions
    - Any use of ++ or --
    - Method invocations
    - Object creation expressions
  - Control Flow statements
    - Selection & repetition structures

Comment Types

- End of line comment – ignores everything else on the line after the "//"

```
// compute the volume
```

- Multi-line comment — must open with "/*" and close with "*/"

```
/*
 * sort the array using
 * selection sort
 */
```

- Javadoc comment — special version of multi-line comment that starts with "/**"
  - Used by Java's documentation tool

```
/**
 * Determines if the item is empty
 * @return true if empty, false otherwise
 */
```

If-Then Statement

- The *if-then* statement is the most basic of all the control flow statements.

Python

```
if x == 2:
    print "x is 2"
print "Finished"
```

Java

```
if (x == 2)
    System.out.println("x is 2");
System.out.println("Finished");
```

Notes about Java's *if-then*:

- Conditional expression must be in parentheses
- Conditional expression must result in a boolean value

Multiple Statements

- What if our *then* case contains multiple statements?

Python

```
if x == 2:
    print "even"
    print "prime"
print "Done!"
```

Java

```
if(x == 2)
    System.out.println("even");
    System.out.println("prime");
System.out.println("Done!");
```

Notes:

- Unlike Python, spacing plays no role in Java's selection/repetition structures
- The Java code is ***syntactically*** fine – no compiler errors
- However, it is ***logically*** incorrect

Blocks

- A ***block*** is a group of zero or more statements that are grouped together by delimiters.
- In Java, blocks are denoted by opening and closing curly braces ‘{’ and ‘}’ .

```
if(x == 2) {
    System.out.println("even");
    System.out.println("prime");
}
System.out.println("Done!");
```

Note:

• It is generally considered a good practice to include the curly braces even for single line statements.

Variable Scope

- That set of code statements in which the variable is known to the compiler.
- Where a variable it can be referenced in your program
- Limited to the code block in which the variable is defined
- For example:

```
if(age >= 18) {
    boolean adult = true;
}
/* couldn't use adult here */
```

If-Then-Else Statement

- The *if-then-else* statement looks much like it does in Python (aside from the parentheses and curly braces).

Python

```
if x % 2 == 1:
    print "odd"
else:
    print "even"
```

Java

```
if(x % 2 == 1) {
    System.out.println("odd");
} else {
    System.out.println("even");
}
```

If-Then-Else If-Then-Else Statement

- Again, very similar…

Python

```
if x < y:
    print "x < y"
elif x > y:
    print "x > y"
else:
    print "x == y"
```

Java

```
if(x < y) {
    System.out.println("x < y");
} else if (x > y) {
    System.out.println("x > y");
} else {
    System.out.println("x == y");
}
```

Switch Statement

- Unlike *if-then* and *if-then-else*, the *switch* statement allows for any number of possible execution paths.
- Works with *byte*, *short*, *char*, and *int* primitive data types.
  - As well as enumerations (which we'll cover later)

Switch Statement

```
int cardValue = /* get value from somewhere */;
switch(cardValue) {
    case 1:
        System.out.println("Ace");
        break;
    case 11:
        System.out.println("Jack");
        break;
    case 12:
        System.out.println("Queen");
        break;
    case 13:
        System.out.println("King");
        break;
    default:
        System.out.println(cardValue);
}
```

Notes:

- *break* statements are typically used to terminate each *case.*
- It is usually a good practice to include a *default* case.

Switch Statement

```
switch (month) {
    case 1: case 3: case 5: case 7:
    case 8: case 10: case 12:
        System.out.println("31 days");
        break;
    case 4: case 6: case 9: case 11:
        System.out.println("30 days");
        break;
    case 2:
        System.out.println("28 or 29 days");
        break;
    default:
        System.err.println("Invalid month!");
        break;
}
```

Note:

- Without a break statement, cases “fall through” to the next statement.

While Loops

- The *while* loop executes a block of statements while a particular condition is *true.*
- Pretty much the same as Python…

Python

```
count = 0;
while(count < 10):
    print count
    count += 1
print "Done!"
```

Java

```
int count = 0;
while(count < 10) {
    System.out.println(count);
    count++;
}
System.out.println("Done!");
```

Do-While Loops

- In addition to *while* loops, Java also provides a *do-while* loop.
  - The conditional expression is at the bottom of the loop.
  - Statements within the block are always executed at least once.
  - Note the trailing semicolon!

```
int count = 0;
do {
    System.out.println(count);
    count++;
} while(count < 10);
System.out.println("Done!");
```

For Loop

- The for statement provides a compact way to iterate over a range of values.

```
for (initialization; termination; increment) {
    /* ... statement(s) ... */
}
```

- The ***initialization expression*** initializes the loop – it is executed once, as the loop begins.
- When the ***termination expression*** evaluates to false, the loop terminates.
- The ***increment expression*** is invoked after each iteration through the loop.

For Loop

- The equivalent loop written as a *for* loop
  - Counting from start value (zero) up to (excluding) some number (10)

Python

```
for count in range(0, 10):
    print count
print "Done!"
```

Java

```
for(int count = 0; count < 10; count++) {
    System.out.println(count);
}
System.out.println("Done!");
```

For Loop

- Counting from 25 up to (excluding) 50 in steps of 5

Python

```
for count in range(25, 50, 5):
    print count
print "Done!"
```

Java

```
for(int count = 25; count < 50; count += 5){
    System.out.println(count);
}
System.out.println("Done!");
```

For Loop

- Iterating over the contents of an array

Python

```
items = ["foo", "bar", "baz"]
for i in range(len(items)):
    print "%d: %s" % (i, items[i])
```

Java

```
String[] items = new String[]{"foo","bar","baz"};
for (int i = 0; i < items.length; i++) {
   System.out.printf("%d: %s%n", i, items[i]);
}
```

For Each Loop

- Java also has a second form of the for loop known as a “for each” or “enhanced for” loop.
- This is much more like Python’s *for-in* loop.
- The general form is:

```
for (<type> <item name> : <collection name>) {
    /* ... do something with item ... */
}
```

- For now, we’ll assume that the collection is an array (though there are other objects it can be, which we’ll discuss later in the semester).

For Each Loop

- Iterating over the contents of an array using a *for-each* loop

Python

```
items = ["foo", "bar", "baz"]
for item in items:
    print item
```

Java

```
String[] items = new String[]{"foo","bar","baz"};
for(String item : items) {
   System.out.println(item);
}
```

Reading From the Console

- Java's ***Scanner object*** reads in input that the user enters on the command line.

```
Scanner input = new Scanner(System.in);
```

- System.in is a reference to the ***standard input buffer.***
- We can read values from the Scanner object using the dot notation to invoke a number of functions.
  - nextInt() — returns the next integer from the buffer
  - nextFloat() — returns the next float from the buffer
  - nextLine() — returns the entire line as a String

Scanner Notes

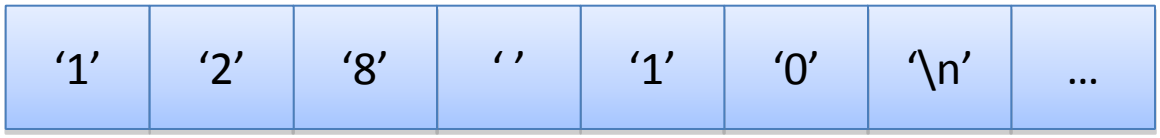
- In order to use the Scanner class, you'll need to add the following line to the top of your code…

```
import java.util.Scanner;
```

- You should ***never*** declare more than one Scanner object on a given input stream.
- The Scanner object will wait for a user to type, and read all text entered up until the user presses the "enter" key (including the newline character).

Reading from the Console

```
System.out.print("Enter 2 numbers to sum: ");
Scanner input = new Scanner(System.in);
int n1 = input.nextInt();
int n2 = input.nextInt();
System.out.printf("%d + %d = %d", n1, n2, n1 + n2);
```



- Let's assume the user has entered "128 10" .
- The first call to nextInt() reads the characters "128" leaving " 10\n" in the input buffer.
- The second call to nextInt() reads the "10" and leaves the "\n" in the buffer.

Reading via UNIX Redirection

```
int sum = 0;
Scanner input = new Scanner(System.in);
while(input.hasNextInt()) {
    sum += input.nextInt();
}
System.out.println("Sum: " + sum);
```

```
% cat numbers
1 2 3
4
5 6 7
8
% java Sum < numbers
Sum: 36
%
```

- The Scanner class also has a bunch of hasNextX() methods to detect if there's another data item of the given type in the stream.
- For example, this is useful if we were reading an unknown quantity of integers from a file that is redirected into our program (as above).

Strings

- Java's String class represents an ***immutable*** sequence of characters.

```
String variable = "ABC";
String name = "Bubba";
```

- Strings can be easily concatenated together using the + operator

```
String player = "Donkey" + "Kong";
```

- Strings can be concatenated with both primitive and reference types.

```
String foo = "abc" + 123;
```

- Strings also support the += operator.

```
String s = "foo";
s += "bar";
```

String Equality

Python

```
if player == "Mario":
    color = "red"
```

Java

```
if(player.equals("Mario")) {
    color = "red";
}
```

- Unlike Python, we cannot simply use the == operator to compare Strings.
- Remember — Strings are reference types, so comparing the variables would simply compare the references.
- Instead, we need to utilize the String class' equals() method.

Strings

- The String class' ***length*** method is used to retrieve the number of characters in a string.

Python

```
print len(name)
```

Java

```
System.out.println(name.length());
```

- To access an individual character of a string, we must use the String class' charAt(index) method.

Python

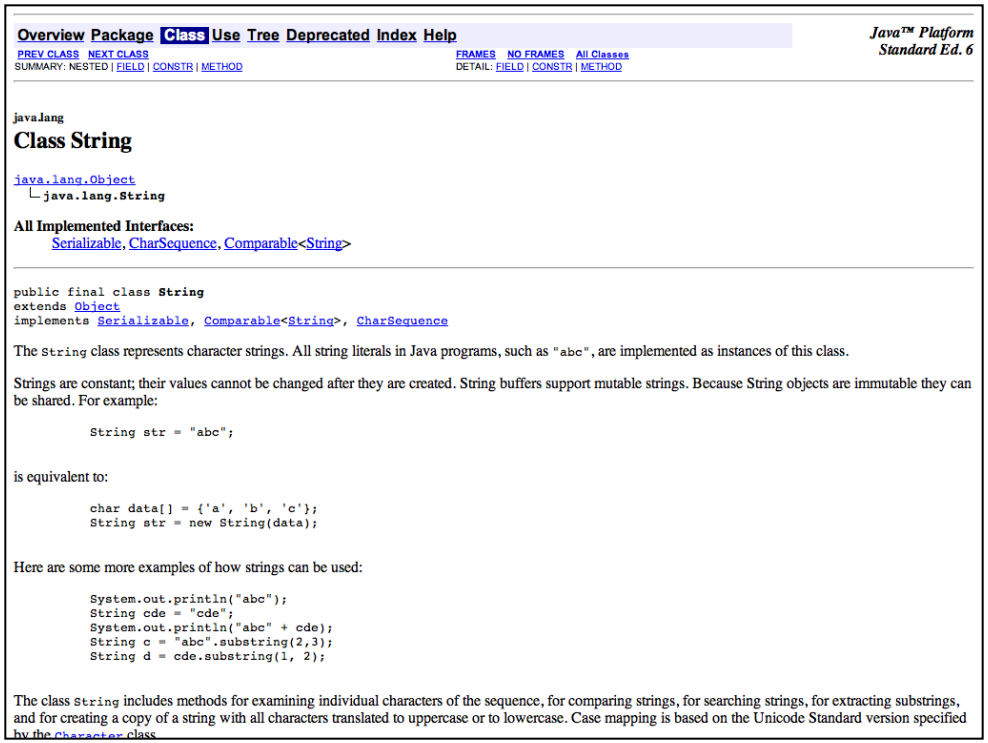
```
player = "Mario"
print "%c" % player[0]
```

Java

```
String player = "Mario";
System.out.println(player.charAt(0));
```

Strings

- To see more String methods, consult the javadocs...
  - http://download.oracle.com/javase/6/docs/api/java/lang/String.html



Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 6

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.lang

Class String

java.lang.Object
  └ java.lang.String

All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.

Java Program Basics

```
package demos;

public class SimpleProgram {
    public static void main (String[] args){
        System.out.println("Hello World");
    }
}
```

- All code (variables, functions, etc.) in Java exist within a class declaration ...
  - Data Structures
  - Driver Classes
- The ***package*** keyword defines a file/class hierarchy used by the compiler and JVM.

Java Program Review

```
package demos;

public class SimpleProgram {
  public static void main (String[] args){
    System.out.println("Hello World");
  }
}
```

```
package demos;

public class OtherProgram {
  public static void main (String[] args){
    System.out.println("Hello World 2");
  }
}
```

- Java source code can be compiled under any operating system.
  - javac -d . SimpleProgram.java
  - javac -d . OtherProgram.java
- Java will create a directory named *demos* containing
  - SimpleProgram.class
  - OtherProgram.class
- We can execute SimpleProgram with the following.
  - java demos.SimpleProgram
- We can execute OtherProgram with the following.
  - Java demos.OtherProgram
- We can execute any class' main in a similar manner.
  - java <package name>.<Class name>

Command Line Arguments

```
package demos;

public class ArgsDemo {
    public static void main (String[] args){
        for(int i = 0; i < args.length; i++){
            System.out.println(args[i]);
        }
    }
}
```

- Anything that follows the name of the main class to be executed will be read as a ***command line argument***.
- All text entered will be stored in the String array specified in main (typically ***args*** by convention).
  - java demos.ArgsDemo Hi
  - Results in “Hi” stored at args[0]
- Individual arguments can be separated by spaces like so
  - java demos.ArgsDemo foo 123 bar
  - Results in “foo” stored at args[0], “123” at args[1] and “bar” at args[2]