CISC 370: Introduction to Java

Instructor: Sara Sprenkle
sprenkle@cis.udel.edu

TA: Ke Li
kli@cis.udel.edu

June 6, 2006

What is Java?
… and, why should I learn it?

What is Java?
… and, why should I learn it?

- From Sun Microsystems
  - 1995, James Gosling and Patrick Naughton
  - Specifications
- Develop cross-platform applications
  - Web, desktop, embedded
- Object-oriented
- Simpler than C++
- Rich and **large** library

What to Expect from this Class

- Programming intensive
    - Interesting assignments and projects
    - More freedom in design
    - Building on large library of classes
- Compare/Contrast with C++
- Learning on your own
    - Online resources

Class Details

- Tuesday, Thursday lectures
  - Quiz at beginning of each Tuesday class
  - See web page for example code, lecture slides
- Expected Participation
- Optional Textbooks
  - Use plentiful online resources instead!

Class Details

- Weekly Programming Assignments
  - Due one week after assigned
- 2 Projects
  - Demos with TA and me
- 1 Exam
  - Final: TBD but August 11ish
- Course Project Manager
  - https://128.4.133.74:8080/scheduler/group.html
  - For viewing grades and scheduling demos

Course Dynamics

- Professor's Responsibilities:
  - Be prepared for class
  - Correct students non-judgmentally
  - Treat students with respect
  - Challenge and encourage students
  - Make class material as clear as possible
- Student's Responsibilities:
  - Be prepared for class (do readings and homework)
  - Give attention and effort in class to learning
  - Ask questions (**during class** and via email)
  - Use professor's and TA’s office hours when you’re having trouble
  - Let professor know if something is going wrong
  - Treat other students, TA, and professor with respect

Survey Says…

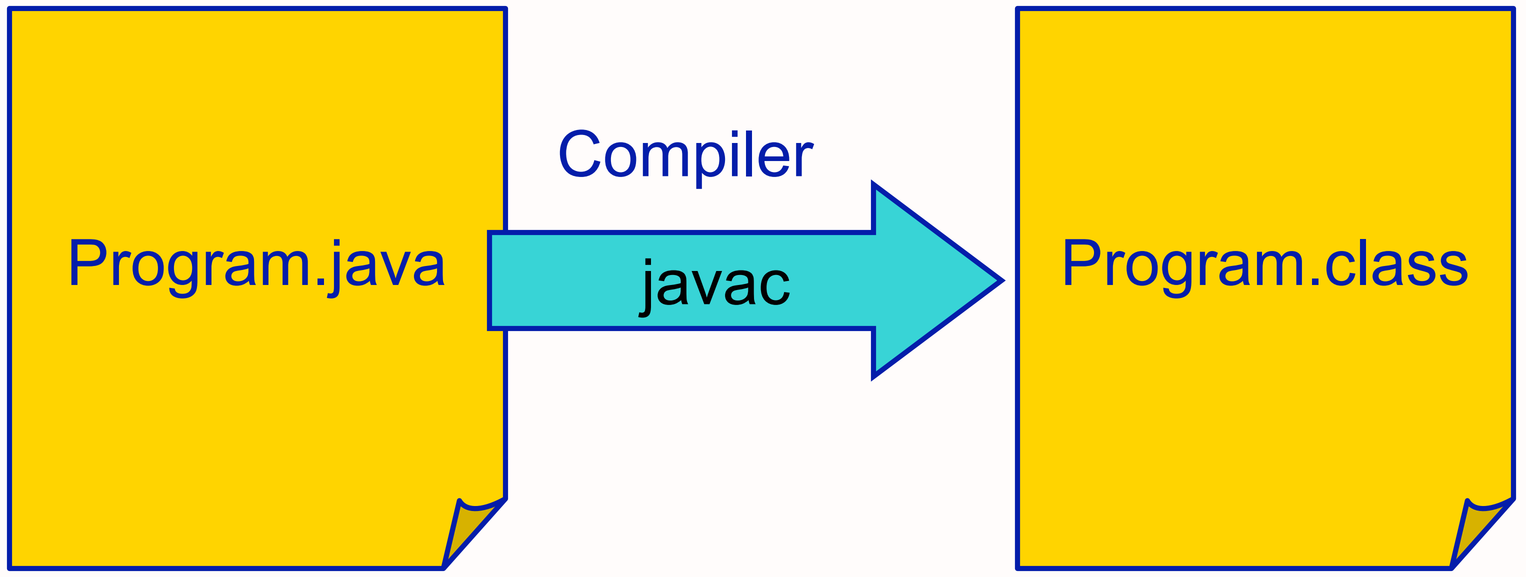
- More about you!

What is Java?

- Java Programming Language
- Java Virtual Machine
- Java Class Libraries

Java Programming Language

- Entirely object-oriented
- Similar to C++



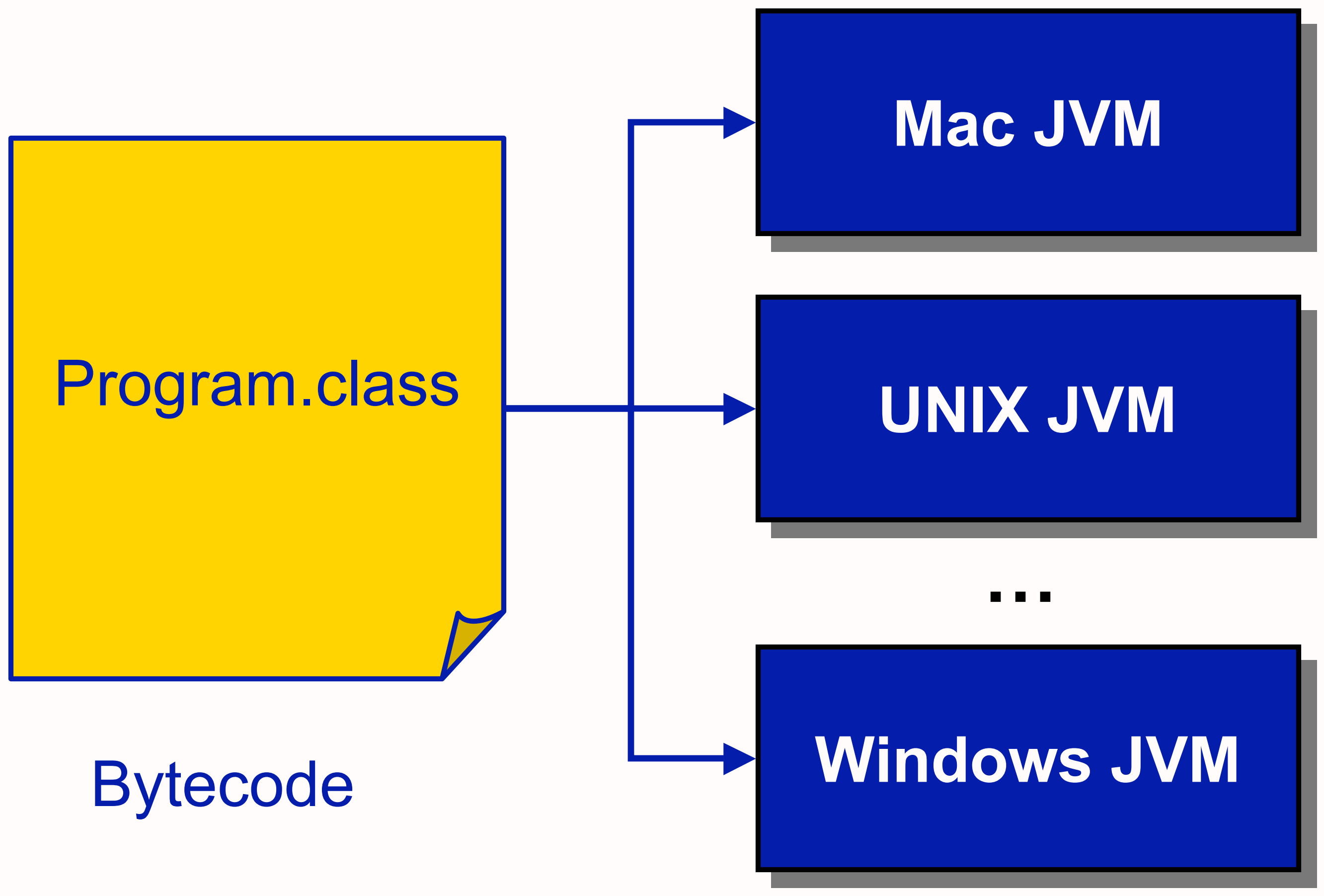
Written in Java Programming Language

Bytecode: machine code for a virtual CPU

Java Virtual Machine (JVM)

- Emulates the CPU, usually specified in software
- Executes the program's bytecode
  - Bytecode: virtual machine code
- Different versions for each platform Java supports
  - program portability

Java Virtual Machine (JVM)



- Same bytecode executes on each platform
  - What happens in C++?

Java Class Libraries

- Pre-defined classes
  - Included with the Java 2 Software Development Kit (SDK) and the Java 2 Runtime Environment (JRE)
  - View the available classes online:

    http://java.sun.com/j2se/1.5.0/docs/api/index.html
- Similar in purpose to library functions included with ANSI C

Benefits of Java

- Rapid development of programs
  - Large library of classes, including GUIs
- Portability
  - run program on multiple platforms without recompiling

Which 'Java' is this class about?

- Java programming language
- Java class libraries

- Use the JVM but won't learn about how it works
  - For more information: http://java.sun.com/docs/books/vmspec/

Java Development Kit (J2SDK)

- J2SDK: Java 2 Software Development Kit
- Free from Sun
- Contains
  - javac: Java compiler
  - java: Java Virtual Machine
  - Java class libraries

Java Development Kit (J2SDK)

- Installed on strauss
  - Java 1.5 should be reachable using default path
  - If not, add `/usr/java1.5/bin` to your path
  - http://www.udel.edu/topics/os/unix/package/java/
- Run `java -version` to determine which version you're running
- You can download the JDK for your machine from http://java.sun.com/j2se/1.5.0/download.jsp
  - JRE is for running Java applications
    - does not include the compiler

Summary of Java Platform SE 5.0

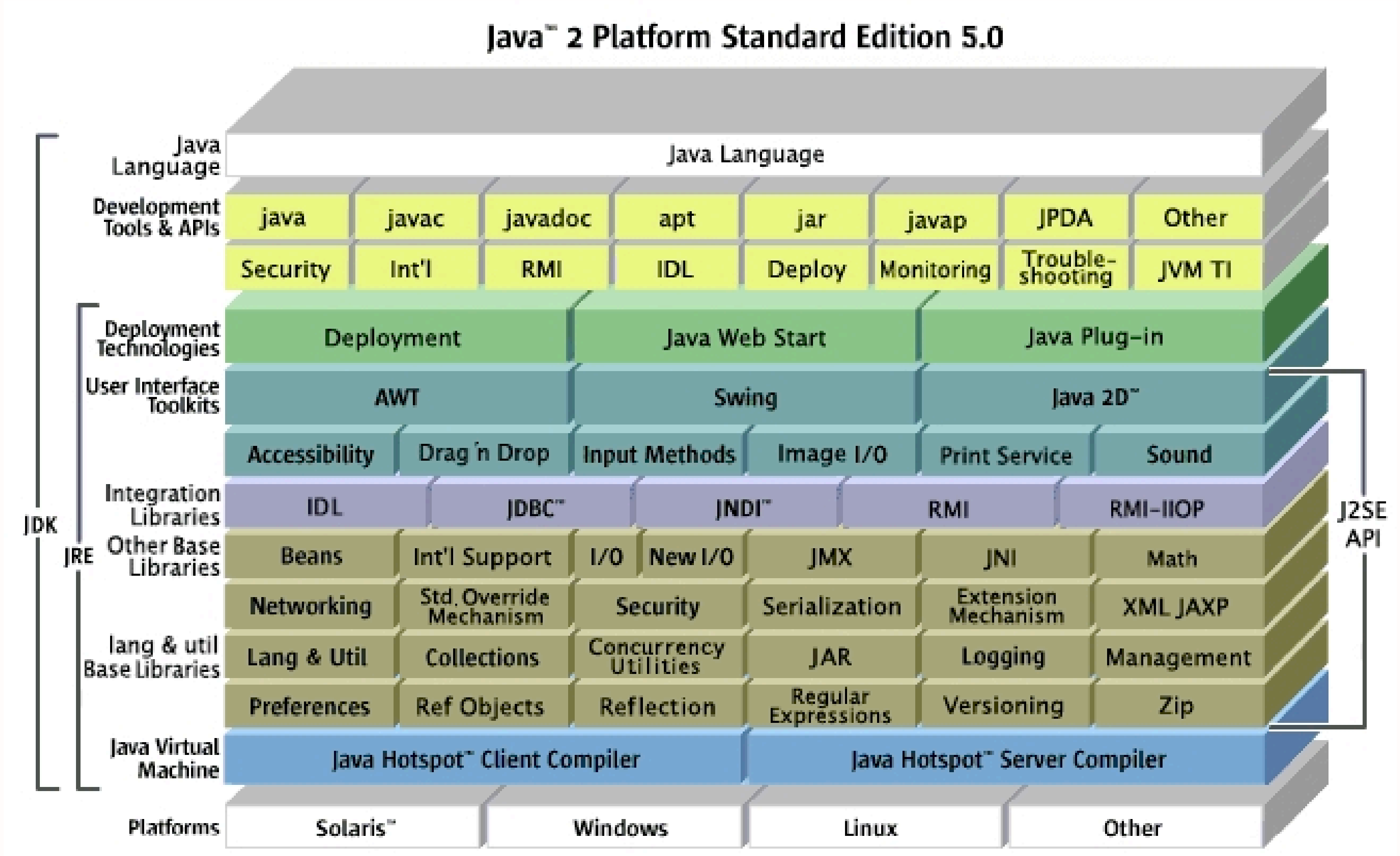


Image from Sun's site

Using the J2SDK

- Compile and run TestProgram.java
  - javac TestProgram.java
    - Compiles the program into TestProgram.class
  - java TestProgram
    - Runs the JVM, which executes the bytecode

Intro to Java Programming Language

- Examples
- Data types
- Control structures

First Java Program

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

First Java Program

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

- Everything in Java is inside a class
  - This class is named "Hello"

First Java Program

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

Access Modifier:
controls if other classes can use code in this class

First Java Program

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

Defines the class "Hello"

First Java Program

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

method

- Class contains one method
    - Functions are called methods in Java

First Java Program

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

- **`main`** methods
  - Similar to **`main`** in C++
  - Takes one parameter: an array of Strings
  - Must be "**`public static`**"
  - Must be void: returns nothing

First Java Program

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

- Method contains one line of code
  - What do you think it does?

First Java Program

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

- Calls the **`println`** method on the **`System.out`** object
- **`println`** takes one parameter, a String
- Displays string on terminal, terminates the line with new line (\n) character

First Java Program

```
/*
Our first Java program
*/
public class Hello
{
     public static void main(String[] args)
     {
          // Print a message
          System.out.println("Hello");
     }
}
```

- Comments: same as C++
  - /* */ or //

Java Fundamentals

Java keywords/reserved words

- Case-sensitive
- Can’t be used for variable or class names
- Many same as in C/C++
- Seen so far …
  - `public`
  - `class`
  - `static`
  - `void`
- Exhaustive list

  http://java.sun.com/docs/books/tutorial/java/nutsandbolts/_keywords.html

Data Types

- Java is *strongly-typed*
  - Every variable must be a declared type
- All data in Java is an *object* except for the primitive data types
  - **int** – 4 bytes (-2,147,483,648 -> 2,147,483,647)
  - **short** – 2 bytes (-32,768 -> 32,767)
  - **long** – 8 bytes (really big integers)
  - **byte** – 1 byte (-128 -> 127)
  - **float** – 4 bytes (floating point)
  - **double** – 8 bytes (floating point)
  - **char** – 2 bytes (Unicode representation)
  - **boolean** – `false` or `true`

Variables

- Declared and initialized same as C and C++
- Typically, names start with lowercase letter
  - '_' also a valid first character
  - Convention: Subsequent words are capitalized
    - Called "Camel Casing"
- Examples:
  - `int x;`
  - `double pi = 3.14;`
  - `char q = 'p';`
  - `boolean isValid = false;`

Camel Casing

More Data Type Information

- Default data types
  - Result of integer division is an int
    - Same as C++
    - Example: `1/2` = ??
- Casting
  - Same as C++ for primitive types
  - Example: `1/(double) 2`

Constants

- Read-only variables
  - Cannot be assigned new value
- Keyword `final` precedes data type

```
final double CM_PER_INCH = 2.540;
```

What was the equivalent keyword in C++?

Class Constants

- Constant variable for all methods in class or for multiple classes
- Much more common than constant instance variables
- Requires `static` keyword
    - static: “for whole class”
    - Also used for methods (will see later)

```
public static final double CM_PER_IN = 2.540;
```

Operators

- Java has most of the same operators as C and C++:
  - +, -, *, /, % (add, subtract, multiple, divide, modulus)
  - ++ and -- (increment and decrement)
  - ==, !=, <, >, <=, >= (relational operators, evaluate to a `boolean` value)
  - &&, ||, ! (logical operators: AND, OR, NOT)
  - &, |, ^, ~ (bitwise AND, OR, XOR, NOT)

Mathematical Functions and Constants

- java.lang.Math class
  - Similar to <math.h>
  - Included in the Java class libraries in the JDK
  - Included by default in every Java program
  - Includes useful mathematical functions (methods) and related constants (`final` values):
    - `double y = Math.pow(x, a);`
    - `double z = Math.sin(y);`
    - `double d = Math.exp(4.59) * Math.PI;`
- Look at java.lang.Math API online

Java API Documentation

- API: Application Programming Interface
  - What the class can do for YOU!
- Complete documentation on every class included with the JDK and on every method and variable contained within each class.

  http://java.sun.com/j2se/1.5.0/docs/api/
- Bookmark it!
  - Too many classes, methods to remember them all
  - Refer to it often

Strings

- Java makes strings very easy, compared to C, C++, and many other languages.
- The Java class libraries include a predefined ‘String’ class in java.lang.String
    - All java.lang classes are automatically included in Java programs

Strings

- Strings are represented by double quotes: “”
- Examples:

```
String emptyString = “”;
String niceGreeting = “Hello there.”;
String badGreeting = “What do you want?”;
```

Note that you don’t need to specify the String’s size

String Concatenation

- Use ‘+’ operator to concatenate Strings

```
String niceGreeting = “Hello”;
String firstName = “Clark”;
String lastName = “Kent”;
String blankSpace = “ ”;

String greeting = niceGreeting + “,” +
     blankSpace + firstName +
     blankSpace + lastName;

System.out.println(greeting);
```

Prints “Hello, Clark Kent”

String Concatenation

- If a string is concatenated with something that is not a string, the other variable is converted to a string.

```
int totalPoints = 110;
int earnedPoints = 87;
float testScore =
  (float)earnedPoints/totalPoints;

System.out.println("Your score is " +
     testScore);
```

String methods: `substring`

- String substring(int beginIndex)
  - Returns a new string that is a substring of this string, from beginIndex through end of this string
- String substring(int beginIndex, int endIndex)
  - Returns a new string that is a substring of this string, from beginIndex to endIndex

```
String greeting = “Hello, Clark Kent!”;
String subStr = greeting.substring(7);
String subStr2 = greeting.substring(7, 11);
```

String methods: `equal`

- boolean equals(Object anObject)
  - Compares this string to the specified object

```
String string1 = "Hello";
String string2 = "hello";
boolean test;
test = string1.equals(string2);
```

- `test` is false because the strings are different

String methods: `equal`

- `string1 == string2` will **not** yield the same result
  - Tests if the objects are the same
  - Not if the contents of the objects are the same

String methods: `equalsIgnoreCase`

- Does what it's named!

```
String string1 = "Hello";
String string2 = "hello";
boolean test;
test = string1.equalsIgnoreCase(string2);
```

- `test` is true!

String methods: `charAt`

- A String is a collection of characters

```
String testString1;
testString1 = "Demonstrate Strings";

char character1;
char character2 = testString1.charAt(3);
character1 = testString1.charAt(2);
```

String methods: and many more!

- boolean endsWith(String suffix)
- boolean startsWith(String prefix)
- length()
- toLowerCase()
- trim(): remove trailing and leading white space
- …
- See Java API for java.lang.String for all

Control Structures

Control Flow: Conditional Statements

- **if** statement

Condition must evaluate to a boolean (booleans are not ints in Java)

```
if (purchaseAmount < availableCredit)
{
   availableCredit = availableCredit -
        purchaseAmount;
   System.out.println("Approved");
}
else
   System.out.println("Denied");
```

Control Flow: Conditional Statements

- **if** statement

```
if (purchaseAmount < availableCredit)
{
    availableCredit = availableCredit -
         purchaseAmount;
    System.out.println("Approved");
}
else
    System.out.println("Denied");
```

Block of code

Blocks of Code

- Everything between { } is a block of code
  - Has an associated scope

Control Flow: Conditional Statements

- **switch** statement

```
int x = 3;
switch(x) {
    case (1) :
            System.out.println("It's a 1.");
            break;
    case (2) :
            System.out.println("It's a 2.");
            break;
    default:
            System.out.println("Not a 1 or 2.");
}
```

Control Flow: while Loops

```
int counter = 0;
while (counter < 10)
{
    System.out.println(counter);
    counter++;
}
System.out.println(“Done.”);
```

Control Flow: do-while loop

- Loop runs at least once

```
int counter = 0;
do {
   System.out.println(counter);
   counter++;
} while (counter < 10);
```

Control Flow: for Loop

```
for (int count=10; count >= 1; count--)
{
    System.out.println("Counting down…" +
            count);
}
System.out.println("Blastoff!");
```

Control Flow: foreach Loop

- New to Java 1.5
  - Sun calls “enhanced for” loop
- Iterate over all elements in an array (or Collection)
- Simple, easy-to-read form

```
int[] a;
int result = 0;
     . . .
for (int i : a)
{
   result += i;
}
```

for each int element `i` in the array `a`

The loop body is visited once for each value of i.

Changing control flow: `break`

- In general, I do **not** recommend using break
  - But, you should know it for reading other people's code
- Exits the current loop

```
while ( <readingdata> ) {
     …
     if( data is null ) { // shouldn't happen
          break;
     }
}
```

Changing control flow: labeled `break`

- Does not exist in C++
- Add a label to a block of code

```
tagged_loop:
while ( <readingdata> ) {
        …
        for ( ) { // not tagged
                if( error condition ) {
                        // get out of both loops
                        break tagged_loop;
                }
        }
}
```

Changing control flow: `continue`

- Jump to the next iteration of the loop

```
while ( <readingdata> ) {
     …
     if( data is null ) { // shouldn't happen
          continue;
     }
     doStuff();
}
```

Alternative way to write code without using continue?

Arrays

- To declare an array of integers:

```
int[] arrayOfInts;
```

  - declaration makes only a variable named `arrayOfInts`
  - does not initialize array or allocate memory for the elements
- To declare *and initialize* array of integers:

```
int[] arrayOfInts = new int[100];
```

Array Initialization

- Initialize an array at its declaration:

```
int [] fibNumbs = {1, 1, 2, 3, 5, 8, 13};
```

  - Note that we do not use the `new` keyword when allocating and initializing an array in this manner

Array Length

- All array variables have a field called `length`

```
int[] array = new int[10];
for (int a = 0; a < array.length; a++)
{
    array[a] = 6;
}
for (int a = 0; a < array.length; a++)
{
    System.out.println(array [a]);
}
```

Overstepping Array Length

- Java safeguards against overstepping length of array
  - Runtime Exception: “Array index out of bounds”
  - More on exceptions later…

```
int[] arrayOfInts = new int[100];
```

  - Attempts to access or write to index < 0 or index >= array.length (100) will generate exception

Array Copying

- It is possible to copy one array variable into another, but then both variables refer to the same array
  - like manipulating pointers in C++

```
int [] fibNumbs = {1, 1, 2, 3, 5, 8, 13};
int [] otherFibNumbs;

otherFibNumbs = fibNumbs;
otherFibNumbs[2] = 99;
```

`fibNumbs[2]` and `otherFibNumbs[2]` are both equal to 99.

```
System.out.println(otherFibNumbs[2]);
System.out.println(fibNumbs[2]);
```

Array Copying

- The copying of an array (element-by-element) can be done using the arraycopy method in the System class

```
System.arraycopy(from, fromIndex, to, toIndex,
  count);
```

- For example:

```
int [] fibNumbs = {1, 1, 2, 3, 5, 8, 13};
int [] otherFibNumbs = new int[7];
System.arraycopy(fibNumbs,0,otherFibNumbs,0,7);
otherFibNumbs[2] = 99;
System.out.println(otherFibNumbs[2]);
System.out.println(fibNumbs[2]);
```

`fibNumbs[2]` = 2,
`otherFibNumbs[2]` = 99

Arrays: Java vs C++

```
int[] array = new int[100]; // Java
```

is not the same as:

```
int array[100];             // C++
```

but is the same as:

```
int * array = new int[100]  // C++
```

- `array` variable is the same as a pointer that points to the beginning of an array in C++.