Topic 2
Java Basics

"On the other hand, Java has already been a big **win** in academic circles, where it has taken the place of **Pascal** as the preferred tool for teaching the basics of good programming…"

-*The New Hacker's Dictionary* version 4.3.1

www.tuxedo.org/~esr/jargon/html/The-Jargon-Lexicon-framed.html

Agenda

- Brief History of Java and overview of language
- Solve a problem to demonstrate Java syntax
- Discuss coding issues and style via example
- Slides include more details on syntax
  - may not cover everything in class, but you are expected to know these

Brief History of Java and Overview of Langauge

java.sun.com/features/1998/05/birthday.html

A brief history of Java

- "Java, whose original name was Oak, was developed as a part of the Green project at Sun. It was started in December '90 by Patrick Naughton, Mike Sheridan and James Gosling and was chartered to spend time trying to figure out what would be the "next wave" of computing and how we might catch it. They came to the conclusion that at least one of the waves was going to be the convergence of digitally controlled consumer devices and computers. "

Applets and Applications

- "The team returned to work up a Java technology-based clone of Mosaic they named "WebRunner" (after the movie *Blade Runner*), later to become officially known as the HotJava™ browser. It was 1994. WebRunner was just a demo, but an impressive one: It brought to life, for the first time, animated, moving objects and dynamic executable content inside a Web browser. That had never been done. [At the TED conference.]"

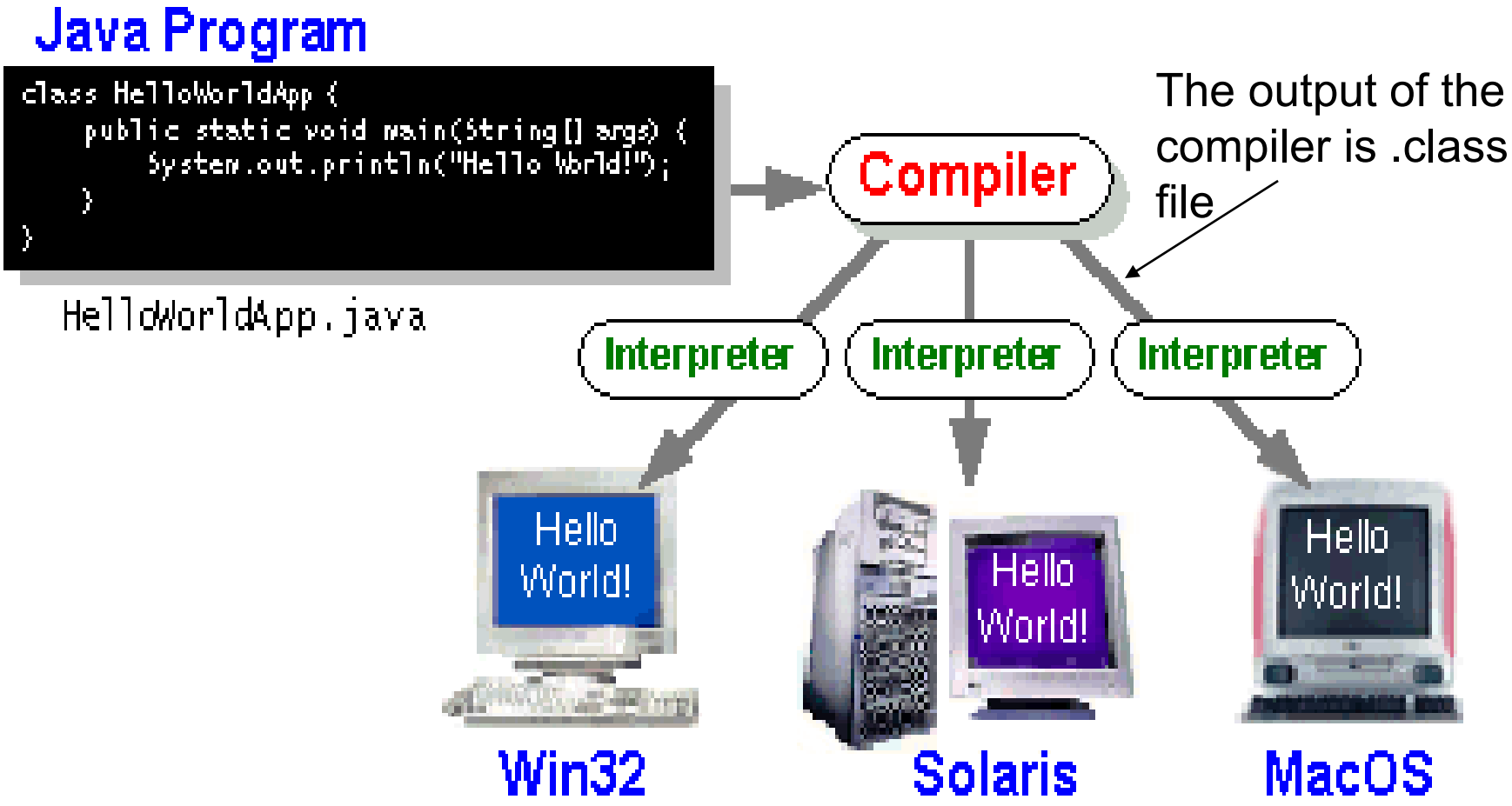
How Java Works

- Java's platform independence is achieved by the use of the *Java Virtual Machine*
- A Java program consists of one or more files with a .java extension
  - these are plain old text files
- When a Java program is compiled the .java files are fed to a compiler which produces a .class file for each .java file
- The .class file contains Java bytecode.
- Bytecode is like machine language, but it is intended for the Java Virtual Machine not a specific chip such as a Pentium or PowerPC chip

More on How Java Works

- To run a Java program the bytecode in a .class file is fed to an interpreter which converts the byte code to machine code for a specific chip (IA-32, PowerPC)
- Some people refer to the interpreter as "The Java Virtual Machine" (JVM)
- The interpreter is platform specific because it takes the platform independent bytecode and produces machine language instructions for a particular chip
- So a Java program could be run an any type of computer that has a JVM written for it.
  - PC, Mac, Unix, Linux, BeaOS, Sparc

A Picture is Worth…



The Interpreter's are sometimes referred to as the Java Virtual Machines

So What!

- The platform independence of Java may be a huge marketing tool, but is actually of little use to people learning Object Oriented Programming and Abstract Data Types
- What is of use is the simplicity of the Java syntax and programming concepts
- Java is a "pure" Object Oriented Language
  - encapsulation, inheritance, and polymorphism
  - all code must be contained in a class
  - no free functions (functions that do not belong to some class) like C++, although someone who wants to write messy Java code certainly can
  - Is OO the best programming paradigm?

HelloWorld.java

```
/**
 * A simple program
 */

public class HelloWorld
{
        public static void main(String[] args)
        {
                System.out.println("HELLO CS307!");
        }
}
```

More on Java Programs

- All code part of some *class*

```
public class Foo
{      //start of class Foo
      /*all code in here!*/
} // end of class Foo
```

- The code for class Foo will be in a file named Foo.java
  - just a text file with the .java extension
  - a class is a programmer defined data type
- A complete program will normally consist of many different classes and thus many different files

Attendance Question 1

What does 6,967 * 7,793 equal?

A. 10,000

B. 23,756,201

C. 54,293,831

D. 2,147,483,647

E. - 2,147,483,648

Attendance Question 2

How many factors does 54,161,329 have?

A. 2

B. 3

C. 4

D. 6

E. more than 6

Bonus question. What are they?

Example Problem

- Determine if a given integer is prime
  - problem definition
  - really naïve algorithm
  - implementation
  - testing
  - a small improvement
  - another improvement
  - yet another improvement
  - always another way ...
  - what about really big numbers? (Discover AKS Primality Testing)

Error Types

- Syntax error / Compile errors
  - caught at compile time.
  - compiler did not understand or compiler does not allow
- Runtime error
  - something “Bad” happens at runtime. Java breaks these into Errors and Exceptions
- Logic Error
  - program compiles and runs, but does not do what you intended or want

Java Language
Review of Basic Features

Basic Features

- Data Types
  - primitives
  - classes / objects
- Expressions and operators
- Control Structures
- Arrays
- Methods
- Programming for correctness
  - pre and post conditions
  - assertions

Java Data Types

Identifiers in Java

- letters, digits, _, and $ (don't use $. Can confuse the runtime system)
- start with letter, _, or $
- by convention:
  1. start with a letter
  2. variables and method names, lowercase with internal words capitalized e.g. honkingBigVariableName
  3. constants all caps with _ between internal words e.g. ANOTHER_HONKING_BIG_INDENTIFIER
  4. classes start with capital letter, internal words capitalized, all other lowercase e.g HonkingLongClassName
- Why? To differentiate identifiers that refer to classes from those that refer to variables

Data Types

- Primitive Data Types
  - byte short int long float double boolean char

```
//dataType identifier;
int x;
int y = 10;
int z, zz;
double a = 12.0;
boolean done = false, prime = true;
char mi = 'D';
```

  - stick with int for integers, double for real numbers
- Classes and Objects
  - pre defined or user defined data types consisting of constructors, methods, and fields (constants and fields (variables) which may be primitives or objects.)

Java Primitive Data Types

Data Type	Characteristics	Range
`byte`	8 bit signed integer	-128 to 127
`short`	16 bit signed integer	-32768 to 32767
`int`	32 bit signed integer	-2,147,483,648 to 2,147,483,647
`long`	64 bit signed integer	-9,223,372,036,854,775,808 to-9,223,372,036,854,775,807
`float`	32 bit floating point number	± 1.4E-45 to ± 3.4028235E+38
`double`	64 bit floating point number	± 4.9E-324 to ± 1.7976931348623157E+308
`boolean`	`true` or `false`	NA, note Java booleans cannot be converted to or from other types
`char`	16 bit, Unicode	Unicode character, `\u0000` to `\uFFFF` Can mix with integer types

What are Classes and Objects?

- Class is synonymous with data type
- Object is like a variable
  - The data type of the Object is some Class
  - referred to as an *instance of a Class*
- Classes contain:
  - the implementation details of the data type
  - and the interface for programmers who just want to use the data type
- Objects are complex variables
  - usually multiple pieces of internal data
  - various behaviors carried out via *methods*

Creating and Using Objects

- Declaration - DataType identifier

```
Rectangle r1;
```

- Creation - new operator and specified constructor

```
r1 = new Rectangle();
Rectangle r2 = new Rectangle();
```

- Behavior - via the dot operator

```
r2.setSize(10, 20);
String s2 = r2.toString();
```

- Refer to documentation for available behaviors (methods)

Built in Classes

- Java has a large built in library of classes with lots of useful methods
- Ones you should become familiar with quickly
- String
- Math
- Integer, Character, Double
- System
- Arrays
- Scanner
- File
- Object
- Random
- Look at the Java API page

import

- import is a reserved word
- packages and classes can be imported to another class
- does not actually import the code (unlike the C++ `include` preprocessor command)
- statement outside the class block

```
import java.util.ArrayList;
import java.awt.Rectangle;
public class Foo{
   // code for class Foo
}
```

More on import

- can include a whole package
  - `import java.util.*;`
- or list a given class
  - `import java.util.Random;`
- instructs the compiler to look in the package for types it can't find defined locally
- the `java.lang.*` package is automatically imported to all other classes.
- Not required to import classes that are part of the same project in Eclipse

The String Class

- String is a standard Java class
  - a whole host of behaviors via methods
- also special (because it used so much)
  - String literals exist (no other class has literals)

```
String name = "Mike D.";
```

  - String concatenation through the + operator

```
String firstName = "Mike";
String lastName = "Scott";
String wholeName = firstName + lastName;
```

  - Any primitive or object on other side of + operator from a String automatically converted to String

Standard Output

- To print to standard output use

  System.out.print(expression); // no newline

  System.out.println(expression); // newline

  System.out.println( ); // just a newline

  common idiom is to build up expression to be printed out

  System.out.println("x is: " + x + " y is: " + y);

Constants

- Literal constants - "the way you specify values that are not computed and recomputed, but remain, well, constant for the life of a program."
  - true, false, null, 'c', "C++", 12, -12, 12.12345
- Named constants
  - use the keyword **`final`** to specify a constant
  - scope may be local to a method or to a class
- By convention any numerical constant besides -1, 0, 1, or 2 requires a named constant

```
final int NUM_SECTIONS = 3;
```

Expressions and Operators

Operators

- Basic Assignment: =
- Arithmetic Operators: +, -, *, /, %(remainder)
  - integer, floating point, and mixed arithmetic and expressions
- Assignment Operators: +=, -=, *=, /=, %=
- increment and decrement operators: ++, --
  - prefix and postfix.
  - avoid use inside expressions.

```
int x = 3;
x++;
```

Expressions

- Expressions are evaluated based on the precedence of operators
- Java will automatically convert numerical primitive data types but results are sometimes surprising
  - take care when mixing integer and floating point numbers in expressions
- The meaning of an operator is determined by its operands

/

is it integer division or floating point division?

Casting

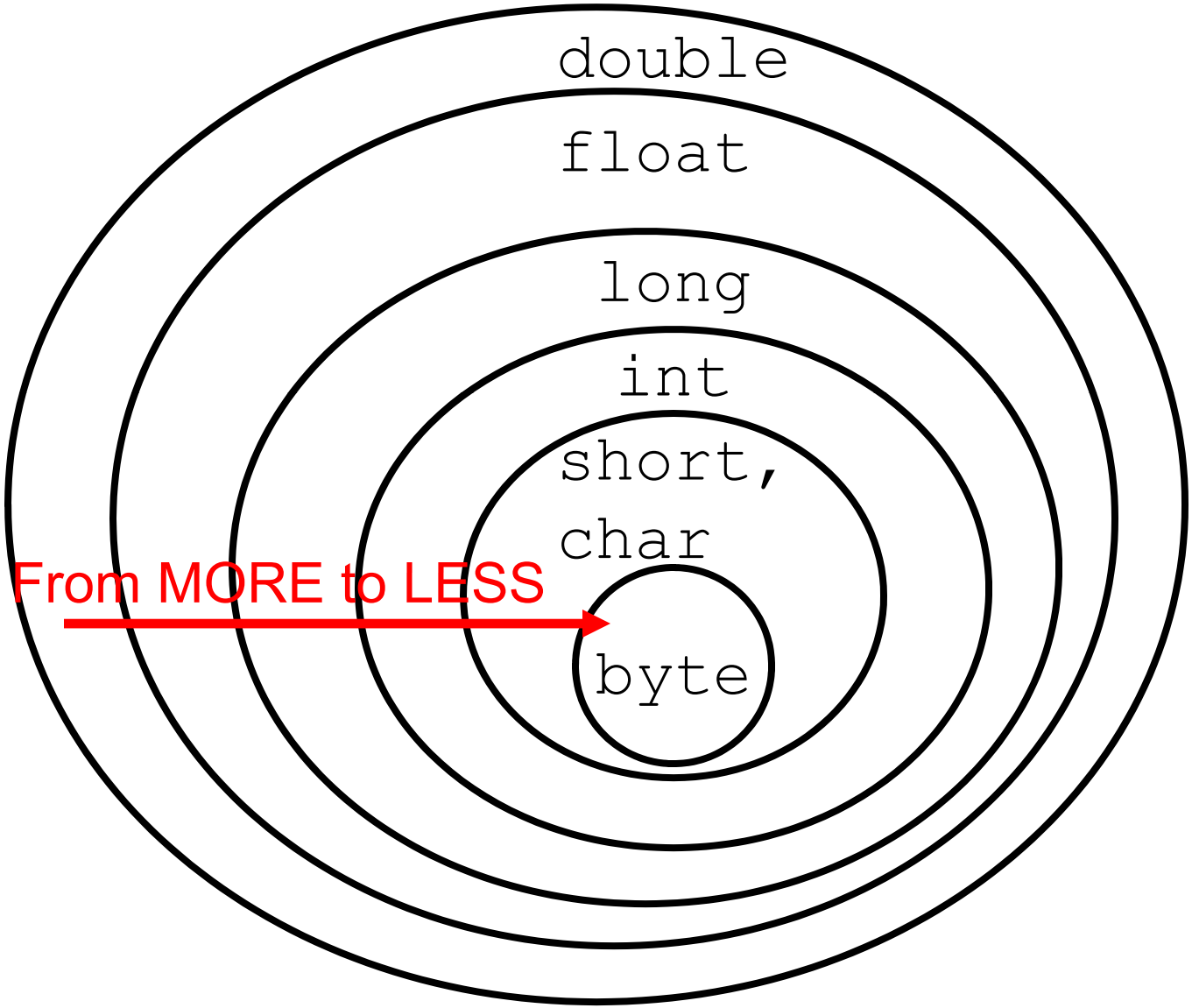
- *Casting* is the temporary conversion of a variable from its original data type to some other data type.
  - *Like being cast for a part in a play or movie*
- With primitive data types if a cast is necessary from a less inclusive data type to a more inclusive data type it is done automatically.

```
int x = 5;
double a = 3.5;
double b = a * x + a / x;
double c = x / 2;
```

- if a cast is necessary from a more inclusive to a less inclusive data type the class must be done explicitly by the programmer
  - failure to do so results in a compile error.

```
double a = 3.5, b = 2.7;
int y = (int) a / (int) b;
y = (int)( a / b );
y = (int) a / b; //syntax error
```

Primitive Casting



Outer ring is most inclusive data type. Inner ring is least inclusive.

In expressions variables and sub expressions of less inclusive data types are automatically cast to more inclusive.

If trying to place expression that is more inclusive into variable that is less inclusive, explicit cast must be performed.

Java Control Structures

Control Structures

- linear flow of control
  - statements executed in consecutive order
- Decision making with if - else statements

```
if(boolean-expression)
   statement;
if(boolean-expression)
{  statement1;
   statement2;
   statement3;
}
```

A single statement could be replaced by a statement block, braces with 0 or more statements inside

Boolean Expressions

- boolean expressions evaluate to true or false
- Relational Operators: >, >=, <, <=, ==, !=
- Logical Operators: &&, ||, !
  - && and || cause short circuit evaluation
  - if the first part of `p && q` is false then q is not evaluated
  - if the first part of `p || q` is true then q is not evaluated

```
//example
if( x <= X_LIMIT && y <= Y_LIMIT)
   //do something
```

More Flow of Control

- if-else:

```
if(boolean-expression)
    statement1;
else
    statement2;
```

- multiway selection:

```
if(boolean-expression1)
    statement1;
else if(boolean-expression2)
    statement2;
else
    statement3;
```

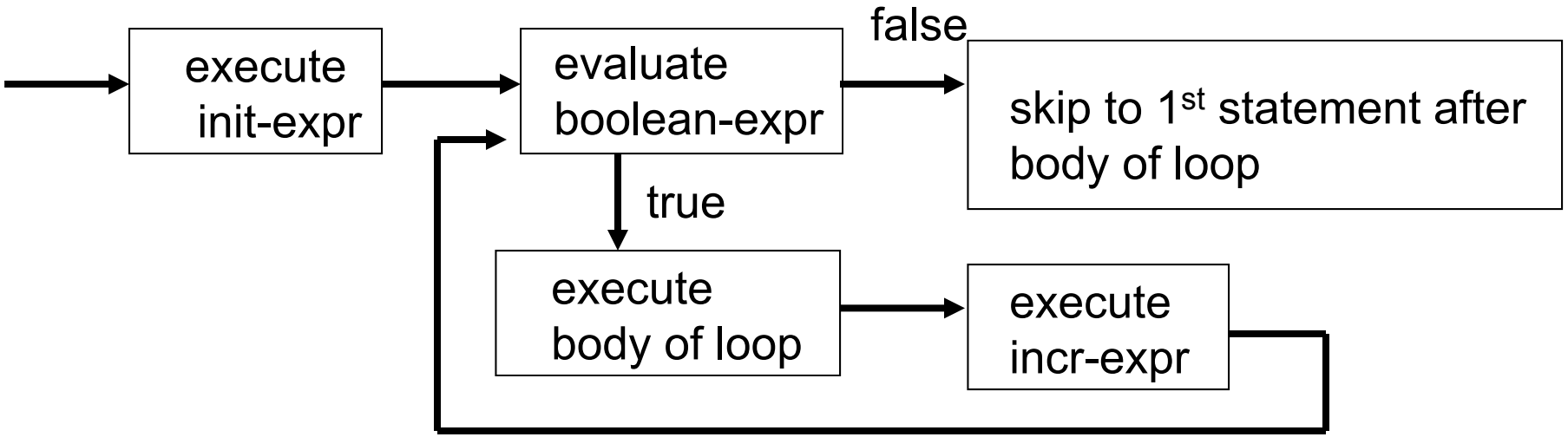
- individual statements could be replaced by a statement block, a set of braces with 0 or more statements
- Java also has the switch statement, but not part of our subset

for Loops

- for loops

```
for(init-expr;boolean-expr;incr-expr)
    statement;
```

- init-expr and incr-expr can be more zero or more expressions or statements separated by commas
- statement could be replaced by a statement block



while loops

- while loops

```
while(boolean-expression)
   statement; //or statement block
```

- do-while loop part of language

```
do
   statement;
while(boolean-expression);
```

- Again, could use a statement block
- `break`, `continue`, and labeled breaks
  - referred to in the Java tutorial as *branching statements*
  - keywords to override normal loop logic
  - use them judiciously (which means not much)

Attendance Question 3

True or false: Strings are a primitive data type in Java.

A. TRUE

B. FALSE

Attendance Question 4

What is output by the following Java code?

```
int x = 3;
double a = x / 2 + 3.5;
System.out.println(a);
```

A. a

B. 5

C. 4.5

D. 4

E. 5.0

Arrays

Arrays in Java

- "Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration. "
  - *S. Kelly-Bootle*
- Java has built in arrays. a.k.a. native arrays
- arrays hold elements of the same type
  - primitive data types or classes
  - space for array must be dynamically allocated with new operator. (Size is any *integer expression*. Due to dynamic allocation does not have to be constant.)

```
public void arrayExamples()
{     int[] intList = new int[10];
      for(int i = 0; i < intList.length; i++)
      {     assert 0 >= i && i < intList.length;
            intList[i] = i * i * i;
      }
      intList[3] = intList[4] * intList[3];
}
```

Array Details

- all arrays must be dynamically allocated
- arrays have a public, final field called *length*
  - built in size field, no separate variable needed
  - don't confuse length (capacity) with elements in use
- elements start with an index of zero, last index is length - 1
- trying to access a non existent element results in an ArrayIndexOutOfBoundsException (AIOBE)

Array Initialization

- Array variables are object variables
- They hold the memory address of an array object
- The array must be dynamically allocated
- All values in the array are initialized (0, 0.0, char 0, false, or null)
- Arrays may be initialized with an initializer list:

```
int[] intList = {2, 3, 5, 7, 11, 13};
double[] dList = {12.12, 0.12, 45.3};
String[] sList = {"Olivia", "Kelly", "Isabelle"};
```

Arrays of objects

- A native array of objects is actually a native array of *object variables*
  - all object variables in Java are really what?
  - Pointers!

```
public void objectArrayExamples()
{       Rectangle[] rectList = new Rectangle[10];
        // How many Rectangle objects exist?

        rectList[5].setSize(5,10);
        //uh oh!

        for(int i = 0; i < rectList.length; i++)
        {       rectList[i] = new Rectangle();
        }

        rectList[3].setSize(100,200);
}
```

Array Utilities

- In the `Arrays` class, static methods
- `binarySearch`, `equals`, `fill`, and `sort` methods for arrays of all primitive types (except boolean) and arrays of Objects
  - overloaded versions of these methods for various data types
- In the System class there is an `arraycopy` method to copy elements from a specified part of one array to another
  - can be used for arrays of primitives or arrays of objects

The `arraycopy` method

- `static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`

  Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.

```
int[] list = new int[10];
// code to fill list
// list needs to be resized
int[] temp = new int[list.length * 2];
System.arraycopy(list, 0, temp, 0,
        list.length);
list = temp;
```

2D Arrays in Java

- Arrays with multiple dimensions may be declared and used

```
int[][] mat = new int[3][4];
```

- the number of pairs of square brackets indicates the dimension of the array.
- by convention, in a 2D array the first number indicates the row and the second the column
- Java multiple dimensional arrays are handles differently than in many other programming languages.

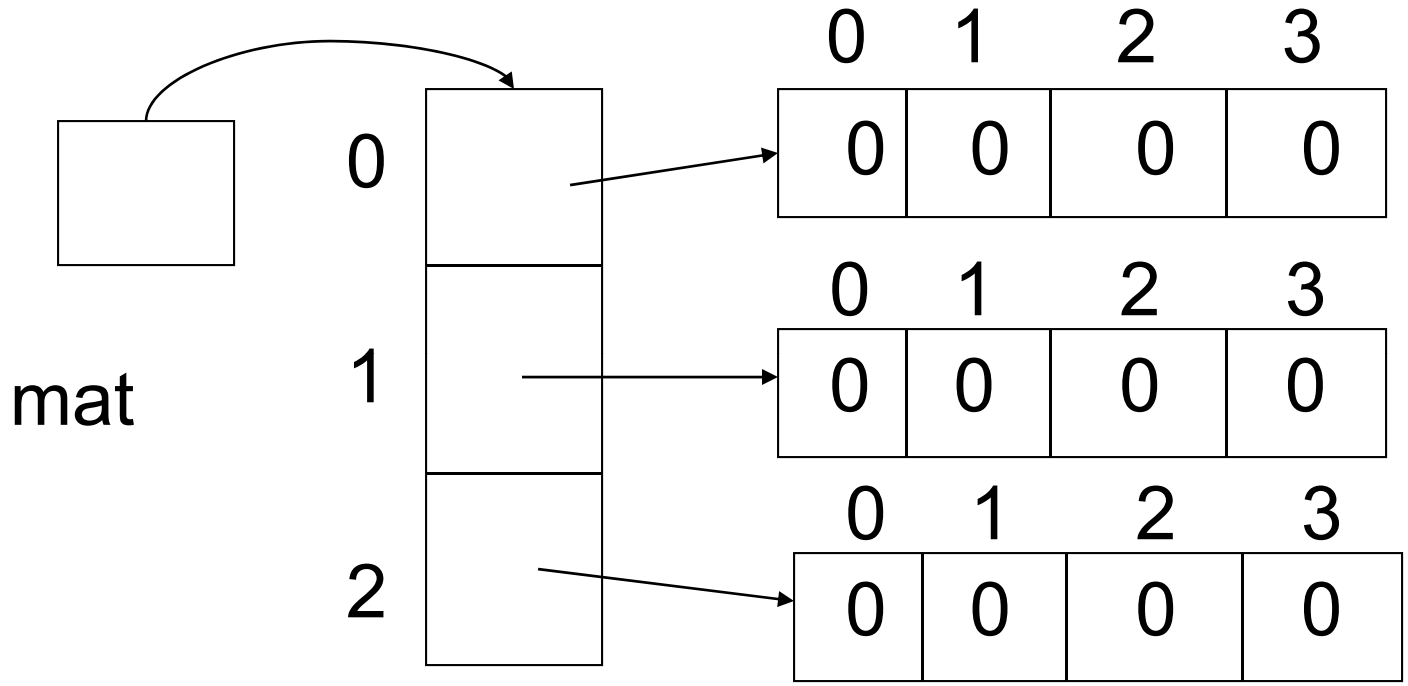
Two Dimensional Arrays

row \ column	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

This is our abstract picture of the 2D array and treating it this way is fine.

```
mat[2][1] = 12;
```

The Real Picture



`mat` holds the memory address of an array with 3 elements. Each element holds the memory address of an array of 4 `int`s

Arrays of Multiple Dimension

- because multiple dimensional arrays are treated as arrays of arrays of arrays……multiple dimensional arrays can be *ragged*
  - each row does not have to have the same number of columns

```
int[][] raggedMat = new int[5][];
for(int i = 0; i < raggedMat.length; i++)
     raggedMat[i] = new int[i + 1];
```

  - each row array has its own length field

Ragged Arrays

- Ragged arrays are sometime useful, but normally we deal with *rectangular* matrices
  - each row has the same number of columns as every other row
  - use this a lot as precondition to methods that work on matrices
- working on matrices normally requires nested loops
  - why is this so hard?

Enhanced for loop

- New in Java 5.0
- a.k.a. the for-each loop
- useful short hand for accessing all elements in an array (or other types of structures) if no need to alter values
- alternative for iterating through a set of values

```
for(Type loop-variable : set-expression)
    statement
```

- logic error (not a syntax error) if try to modify an element in array via enhanced for loop

Enhanced for loop

```
public static int sumListOld(int[] list)
{     int total = 0;
      for(int i = 0; i < list.length; i++)
      {     total += list[i];
            System.out.println( list[i] );
      }
      return total;
}
```

```
public static int sumListEnhanced(int[] list)
{     int total = 0;
      for(int val : list)
      {     total += val;
            System.out.println( val );
      }
      return total;
}
```

Attendance Question 5

What is output by the code to the right when method d1 is called?

A. 322

B. 323

C. 363

D. 366

E. 399

```
public void d2(int x){
   x *= 2;
   System.out.print(x);
}

public void d1(){
   int x = 3;
   System.out.print(x);
   d2(x);
   System.out.print(x);
}
```

Attendance Question 6

What is output by the code to the right?

```
int[] list = {5, 1, 7, 3};
System.out.print( list[2] );
System.out.print( list[4] );
```

A. Output will vary from one run of program to next

B. 00

C. 363

D. 7 then a runtime error

E. No output due to syntax error

Methods

Methods

- methods are analogous to procedures and functions in other languages
  - local variables, parameters, *instance variables*
  - must be comfortable with variable scope: where is a variable defined?
- methods are the means by which objects are manipulated (objects *state* is changed) - much more on this later
- method header consists of
  - access modifier(**public**, package, protected, **private**)
  - static keyword (optional, class method)
  - return type (void or any data type, primitive or class)
  - method name
  - parameter signature

More on Methods

- local variables can be declared within methods.
  - Their scope is from the point of declaration until the end of the methods, unless declared inside a smaller block like a loop
- methods contain statements
- methods can call other methods
  - in the same class: `foo();`
  - methods to perform an operation on an object that is in scope within the method: `obj.foo();`
  - static methods in other classes:

```
double x = Math.sqrt(1000);
```

static methods

- the main method is where a stand alone Java program normally begins execution
- common compile error, trying to call a non static method from a static one

```
public class StaticExample
{   public static void main(String[] args)
    {      //starting point of execution
           System.out.println("In main method");
           method1();
           method2(); //compile error;
    }


    public static void method1()
    {      System.out.println( "method 1");     }


    public void method2()
    {      System.out.println( "method 2");     }
}
```

Method Overloading and Return

- a class may have multiple methods with the same name as long as the parameter signature is unique
  - may not overload on return type
- methods in different classes may have same name and signature
  - this is a type of polymorphism, not method overloading
- if a method has a return value other than void it must have a return statement with a variable or expression of the proper type
- multiple return statements allowed, the first one encountered is executed and method ends
  - style considerations

Method Parameters

- a method may have any number of parameters
- each parameter listed separately
- no VAR (Pascal), &, or const & (C++)
- final can be applied, but special meaning
- all parameters are pass by value
- Implications of pass by value???

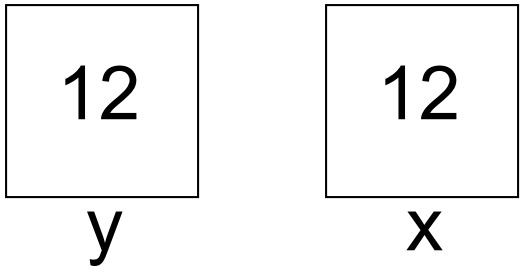
Value Parameters vs. Reference Parameters

- A value parameter makes a copy of the argument it is sent.
  - Changes to parameter do not affect the argument.
- A reference parameter is just another name for the argument it is sent.
  - changes to the parameter are really changes to the argument and thus are permanent

Value vs. Reference

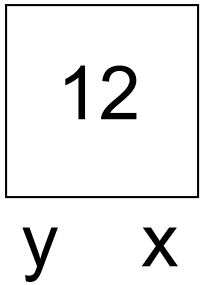
```
// value
void add10(int x)
{    x += 10;      }

void calls()
{    int y = 12;
     add10(y);
     // y = ?
}
```




```
// C++, reference
void add10(int& x)
{    x += 10;      }

void calls()
{    int y = 12;
     add10(y);
     // y = ?
}
```



Programming for Correctness

Creating Correct Programs

- methods should include *pre conditions* and *post conditions*
- Preconditions are things that must be true before a method is called
- Postconditions are things that will be true after a method is complete if the preconditions were met
- it is the responsibility of the caller of a method to ensure the preconditions are met
  - the class must provide a way of ensuring the precondition is true
  - the preconditions must be stated in terms of the interface, not the implementation
- it is the responsibility of the class (supplier, server) to ensure the postconditions are met

Programming by Contract

- preconditions and postconditions create a contract between the client (class or object user) and a supplier (the class or object itself)
  - example of a contract between you and me for a test

	Obligations	Benefits
Client (Student)	*(Must ensure preconditions)* Be at test on time, bring pencil and eraser, write legibly, answer questions in space provided	*(May benefit from postcondition)* Receive fair and accurate evaluation of test to help formulate progress in course
Supplier (Mike)	*(Must ensure postcondition)* Fairly and accurately grade test based on universal guidelines applied to all tests	*(May assume preconditions)* No need to grade test or questions that are illegible, on wrong part of exam, or give makeup exams for unexcused absences

Thinking about pre and postconditions

- pre and postconditions are part of design
- coming up with pre and postconditions at the time of implementation is too late
- the pre and post conditions drive the implementation and so must exist before the implementation can start
  - The sooner you start to code, the longer your program will take.

    -Roy Carlson, U Wisconsin
- *You must spend time on design*

Precondition Example

```
/**
* Find all indices in <tt>source</tt> that are the start of a complete
* match of <tt>target</tt>.
* @param source != null, source.length() > 0
* @param target != null, target.length() > 0
* @return an ArrayList that contains the indices in source that are the
* start of a complete match of target. The indices are stored in
* ascending order in the ArrayList
*/
public static ArrayList<Integer> matches(String source, String target) {
        // check preconditions
        assert (source != null) && (source.length() > 0)
                && (target != null) && (target.length() > 0)
                : "matches: violation of precondition";
```

Creating Correct Programs

- Java features has a mechanism to check the correctness of your program called *assertions*
- Assertions are statements that are executed as normal statements if assertion checking is on
  - you should always have assertion checking on when writing and running your programs
- Assertions are boolean expressions that are evaluated when reached. If they evaluate to true the program continues, if they evaluate to false then the program halts
- logical statements about the condition or state of your program

Assertions

- Assertions have the form

```
assert boolean expression : what to output
  if assertion is false
```

- Example

```
if ( (x < 0) || (y < 0) )
{ // we know either x or y is < 0
  assert x < 0 || y < 0 : x + " " + y;
  x += y;
}
else
{ // we know both x and y are not less than zero
  assert x >= 0 && y >= 0 : x + " " + y;
  y += x;
}
```

- Use assertion liberally in your code
  - part of style guide

Assertions Uncover Errors in Your Logic

```
if ( a < b )
{ // we a is less than b
   assert a < b : a + " " + b;
   System.out.println(a + " is smaller than " + b);
}
else
{ // we know b is less than a
   assert b < a : a + " " + b;
   System.out.println(b + " is smaller than " + a);
}
```

- Use assertions in code that other programmers are going to use.
- In the real world this is the majority of your code!

javadoc

- javadoc is a program that takes the comments in Java source code and creates the html documentation pages
- Open up Java source code. (Found in the src.zip file when you download the Java sdk.)
- Basic Format

```
/** Summary sentence for method foo. More details. More
    details.
    pre: list preconditions
    post: list postconditions
    @param x describe what x is
    @param y describe what y is
    @return describe what the method returns
*/
public int foo(int x, double y)
```

- Comments interpreted as html