Java II

Walter Wes
UCSD Exten
Information Technology
Software Enginee

Module 1

- Course Orientation and Introduction
- Objects and Classes
- Types and Casting
- Relational Expressions
- Primitives
- Strings
- Variables and Memory Maps
- Class and Object Methods
- The CLASSPATH
- Defining Classes and Creating Objects
- Sending Messages to Objects
- FileChooser and Picture
- Playing a Sound
- Object References vs. Primitives
- Backslashes and Slashes

Course Orientation

- Communication
  - Email: All email should be sent via the Blackboard email facility.
  - Contact by email is preferred, however, if and when it is necessary the instructor can be contacted by phone (see syllabus).
  - All projects will be posted on the Blackboard web site. See syllabus for schedule.
  - Students are responsible for periodically logging on to the Blackboard web site.
  - Completed projects must be submitted via the Blackboard web site.
  - The instructor will not be able to rigorously debug student projects. However, if a compile-error has a student stumped, then the error message and the associated source code should be copy-and-pasted and posted on the course discussion list.
  - Lecture slides will cover the course topics in roughly the same order as presented in the textbook. Refer to the syllabus for a mapping of course modules to textbook chapters.
  - Unless explicitly stated otherwise, quotes within the slides are taken from

    Introduction to Computing & Programming with Java, by Mark Guzdial and Barbara Ericson, Pearson Prentice Hall, 2007, ISBN: 0131496980

Course Orientation

- Self-Motivation
  - It is imperative that students are self-motivated. This course requires a great deal of work over a short period of time. ***Be careful not to fall behind!***
  - There will be 4 projects.
  - Students are expected to complete all projects and submit them on time.
  - For more detailed descriptions of the projects, see the *Assignments* page of the our Blackboard course web site.
  - There will be 5 quizzes, one for each module.

Introduction

- A *program* or *algorithm* is like a *recipe*.
- Fundamentally, computers are *encoding/decoding machines*.
- Everything that a computer represents, whether it be text, numbers, graphics, sound, metadata, or the signal radio source SHGb02+14a, is an encoding of a series of ones and zeroes.
- The ones and zeroes within a computer are physically rendered as voltage levels on wires and electro-magnetic fluctuations within memory.
- Encoding schemes can be layered on top of encoding schemes on top of...

Introduction

- *Moore's Law* – The number of transistors on a chip doubles every two years.
- Although its imminent demise has been repeatedly predicted, the law *has held for over 40 years*.
- The doubling period was initially declared to be 12 months—but progress slowed, so the period was adjusted to 18 months (as stated in our textbook), and eventually extended to 2 years.
- So, although the doubling period has changed a little, the law continues to hold steadily. ;-)

Introduction

- Gudzial and Ericson, our textbook authors, cite an interesting statement made by Alan Perlis, the first recipient of the ACM Turing Award.
- Specifically, Gudzial and Ericson write:

  "In 1961, Alan Perlis gave a talk at MIT where he made the argument that computer science, and programming explicitly, should be part of a general, liberal education."
- Gudzial and Ericson then compare programming with calculus, a subject that many people consider to be a mandatory part of a well-rounded, liberal education; they write:

  "Calculus is the study of rates, which is important in many fields[...] Computer science[...] is the study of *process*. Process is important to nearly every field, from business to science to medicine to law. Knowing process formally is important to everyone."

Introduction

- I like to tell students that...

  Software is nothing but applied metaphysics!
- You create your own virtual worlds, within which you are the God that defines all of the rules.
- You ask and answer all of the same existential questions:
  - What is Identity?
  - What is an object?
  - How and when are objects created and destroyed?
  - What do my objects mean and why?
  - How do my objects discover each other?
  - What are the abstractions that are shared by my objects?
  - Et cetera.

Introduction

- In the preface of our textbook, the authors state:

  > One of the lessons from the research on computing education is that one doesn't just "learn to program." One learns to program *something*. How motivating that *something* is can make the difference between learning to program or not.

- As we step through the textbook, you will be learning about a great deal about media and how to manipulate it programmatically. Playing with media is a fun premise for learning how to program in Java.

Introduction

- *Emergence* is fascinating phenomenon that spontaneously manifests within nature as well as within complex software systems.
- The ramifications of the emergence phenomenom may tell us a great about *how consciousness has evolved*, and may help us learn how to create more interesting virtual worlds.
- Recommended Reading
  - At the end of Chapter 1, our textbook recommends the following books for extra reading:
    - *Chaos*, by James Gleick.
    - *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*, by Mitchel Resnick.
    - *Exploring the Digital Domain*, by Ken Abernethy and Tom Allen.
  - I also recommend the following book:
    - *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*, by Steven Johnson.

Introduction

- DrJava – Your Principal IDE
  - Developing software with Java is facillitated through use of an Integrated Development Environment (IDE).
  - Many excellent Java IDEs are available, some of them freely available on the web.
  - Some recommended IDEs included:
    - DrJava
    - Eclipse
    - JCreater
    - NetBeans
    - ItelliJIDEA
    - JBuilder
  - For this course, it is recommended that you use DrJava as your principal IDE.

Introduction

- Why DrJava?
  - DrJava has what is called an *interactions pane*. This pane allows you to issues Java statements that are executed immediately as part of an interactive dialog with an underlying Java interpreter.
  - The immediate feedback provided by the interactions pane is an extremely powerful learning aid.
  - The immediate feedback also helps to address, as least in part, a common problem with online courses—namely, *how do students get feedback so that they know they are on track?*

Introduction

- Additionally, our textbook is written with the assumption that you will take the statements provided to you, and enter them into the interactions pane of your DrJava IDE.
- *As you read* the textbook, *you should be doing* at the same time.
- Lastly, using DrJava is fun!

Introduction

- Do I have to use DrJava?
  - If you are an experienced programmer, and you feel more comfortable with another IDE, please feel free to use your preferred IDE to complete on your course assignments.
  - Regardless, please install and use DrJava as a *read and do tool* to help you study the course material—you will get much more out of the course that way.
  - So, yes, in order to get the most out of this course, you must use DrJava!

Introduction

- How do I install DrJava?
  - DrJava comes with the CD at the back of the textbook.
  - *Do not use the DrJava on the CD!*
  - The version on the CD might be old, and have bugs that have been fixed on more recent versions.
  - Since this is an online course, you have web access and can go to the DrJava web site for the *current stable release*:

    www.drjava.org
  - The DrJava web site provides a quick-start guide and some video tutorials that explain how to get up and running.
  - Our textbook also provides helpful information.

Classes and Objects

- Classes are Object Factories – with a Class, you can create any number of Objects from it.
- The Class defines the details of the structure and the behavior associated with the Objects that it creates.
- One analogy often used to explain the relationship between a Class and the Objects it creates is the process of making cookies.
- To make cookies, one rolls out the cookie dough onto a flat surface. One then takes a cookie cutter and stamps out cookies.
  - The cookie cutter is like a Class.
  - The stamped-out cookies are like Objects created from that Class.

Types and Casting

- What's a *data type*?
  - The concept of a *data type* is fundamental to understanding how to program.
  - There are *two important aspects* to any given data type
    - *Size* – How much memory is set aside (allocated) to be used by the data of that data type?
    - *Encoding Scheme* – How will the bits in memory be interpreted based on that data type?
  - Whereas *size* deals with the implementation details of *how much space* is need to manage the data, the *encoding scheme* deals with the *semantics* (meaning) of the data residing in that space.

Types and Casting

- The *float* data type
  - float answer = 42;

    ...answer is declared to be of data type *float* and is assigned a value of *42* to it.
  - A Java *float* has a size of 32 bits.
  - The bits are interpreted to represent floating point decimal values, comprised of a whole part and a fractional part.
- The *double* data type
  - *Doubles* are essentially the same as *floats*, but they have twice the size, hence their name.
  - A Java *double* has a size of 64 bits, and can therefore store a wider range of numbers with *greater precision*.
- A data type with fewer bits is known as a *narrower* data type, whereas a data type with a greater number of bits is known as a *wider* data type.
- *double* is a *wider* data type than *float*, and *float* is a *narrower* data type than *double*.

Types and Casting

- The *int* data type
  - int answer = 42;

    ...*answer* is declared to be of data type *int*, and is assigned a value of *42*.
  - A Java *int* has a size of 32 bits.
  - The bits are interpreted to represent integral values from -2,147,483,648 to 2,147,483,647 (inclusive).
  - int num = 5;

    42 and *5* are examples of *literal values* (they are literally exactly what they say they are).

Types and Casting

- Using Numeric Literals
  - System.out.println(42 / 5);

    ...displays *8*, since the operands are *int* values, integer division is performed.
  - System.out.println(42.0 / 5.0);

    ...displays *8.4*, since the operands are *float* values, floating point division is performed.
- Casting
  - System.out.println((double) 42 / 5);

    ...displays *8.4*, since *42* is explicitly cast to be a *double*.
  - ((double) 42 / 5) is an example of what is called a *mixed expression*, because the expression is composed of more than one data type. In this case, we have an expression of one *double* (the *42* has been cast as a *double*) and one *int*.
  - When a *mixed numeric expression* is evaluated, the *narrower* data type value (in this case, *5*) is converted to be of the *wider* data type, and the evaluated value will have the size and type of the *wider* data type.

Types and Casting

- More data types
  - The *long* data type
    - long bigNum = 90000000000000000;

      ...*bigNum* is declared to be of data type *long*, and is assigned a value of a really big number!
    - A Java *long* has a size of 64 bits, twice the size of an *int*.
    - The bits are interpreted to represent integral values from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (inclusive).
    - The *long* data type is essentially the same as the *int* data type, except that it supports twice the range of numbers.

Types and Casting

- More data types (continued)
  - The *char* data type
    - char letterGrade = 'A';

      ...*letterGrade* is declared to be of data type *char*, and is assigned a value of *'A'*.
    - The size of a Java *char* is 16 bits.
    - The bits are interpreted to represent integral values that map to the Unicode character set (e.g., the 16 bit encoding of 'A' is 0000000001000001).

Types and Casting

- More data types (continued)
  - The boolean data type
    - boolean done = false;

      ...*done* is declared to be of data type *boolean*, and is assigned a value of *false*.
    - The size of a Java *boolean* is an implementation detail determined by the Java Virtual Machine (JVM) – thus, the size of a *boolean* may vary from one JVM to another.
    - The bits (or bit, as the case may be) are (is) interpreted to represent values of either *true* or *false*.
    - As we will see, logical control structures and repetition structures make use of expressions that evaluate to *true/false boolean* values.

Relational Operators

- Relational Operators are operators that compare two items.
- The result of comparing two items using a relational operator is either *true* or *false*.
- List of relational operators:

Operator	Meaning
“==”	Is Equal To
“!=”	Is Not Equal To
“>”	Is Greater Than
“<”	Is Less Than
“>=”	Is Greater Than or Equal To
“<=”	Is Less Than or Equal To

Relational Operators

- Examples
  - System.out.println(42 **>** 8);

    ...displays *true*, since *42 is greater than 8*.
  - System.out.println(42 **<** 8);

    ...displays *false*, since *42 is not less than 8*.
  - System.out.println('m' **<** 'n');

    ...displays *true*, since *'m' is alphabetically less than 'n'*.
  - System.out.println('p' **!=** 'q');

    ...displays *true*, since *'p' is not equal to 'q'*.
  - System.out.println(42 **==** 42);

    ...displays *true*, since *42 is equal to 42*.
  - Note that expressions can be written so as to be redundant. For example,

    System.out.println((42 > 8) == true);
  - This is not recommended, however. A clearer, more elegant formulation would be:

    System.out.println(42 > 8);

Primitives

- *int*, *long*, *float*, *double*, and *char* are called *primitive data types*, or just *primitives.*
- They are call primitives to indicate that they are rudimentary data types (they store only data), and are *not true objects*.
- It is because of the existence of primitives that Java is not a *pure* object-oriented language.
- In a pure object-oriented language, like C# and Smalltalk, there are no primitives, and everything is an object.

Strings

- A *String* is a sequence of characters.
- As an analogy, envision a piece of twine upon which you thread a series of beads. Think of the beads as being individual characters—the whole strand of characters is a *String*.
- In Java, a *String* is not a *primitive*, it is an *object*.
- *String literals* are denoted through the use of double quotes.
- For example, here are some *String literals*:
  - System.out.println(“Java”);

    ...displays *Java*.
  - System.out.println(“42 / 5”);

    ...displays *42 / 5*.

Strings

- *String Concatenation*
  - Strings can be *concatenated* by using the *concatenation operator*.
  - The *concatenation operator* is the plus sign, “+”, and it operates upon two *String* operands.
  - System.out.println(“Java ” + “rocks!”);

    ...displays *Java rocks!*.
  - You can think of *concatenation* as the way that you “add” *String*s together.
  - I also like to think of *concatenation* as being like gluing two *String* together.
  - Note that what really happens when you *concatenate* two *String*s is that a new *String* is created . That new *String* represents a fusion of the two *String* operands.

Variables and Memory Maps

- We have actually already seen some uses of *variables*.
- *Variables* provide a means of having a name for slot in memory that holds a value of a specified data type.
- *Variables* **allow** you to store the results of your processing.
- *Variables* can be reused again and again, and their associated value *can change* over time—hence, the reason they are called *variables*.
- float numerator = 42;

  ...declares a *variable* named *numerator* of data type *float*, and initializes that *variable* with the value *42*.
- float denominator = 5;

  ...declares a *variable* named *denominator* of data type *float*, and initializes that *variable* with the value *5*.
- System.out.println(*numerator* / *denominator*);

  ...displays *8.4*, since numerator has a *float* value of *42* and denominator has a *float* value of *5*.

Variables and Memory Maps

- float answer;

  ...declares a *variable* named *answer* of data type *float*. Note that in this example, we do not initialize the variable with a value of any kind. The Java compiler will ensure that the variable is initialized with a value of zero, i.e. *0*.

- answer = numerator / denominator;

  ...assigns the calculated value of *8.4* to the variable *answer*.

- denominator = 8;

  ...assigns the value of *8* to the variable denominator.

- System.out.println(answer);

  ...displays *8.4*, which is the value that was stored in the variable *answer*.

- System.out.println(numerator / denominator);

  ...displays *5.25*, which is the result of dividing the current value of *numerator* by the current value of *denominator*.

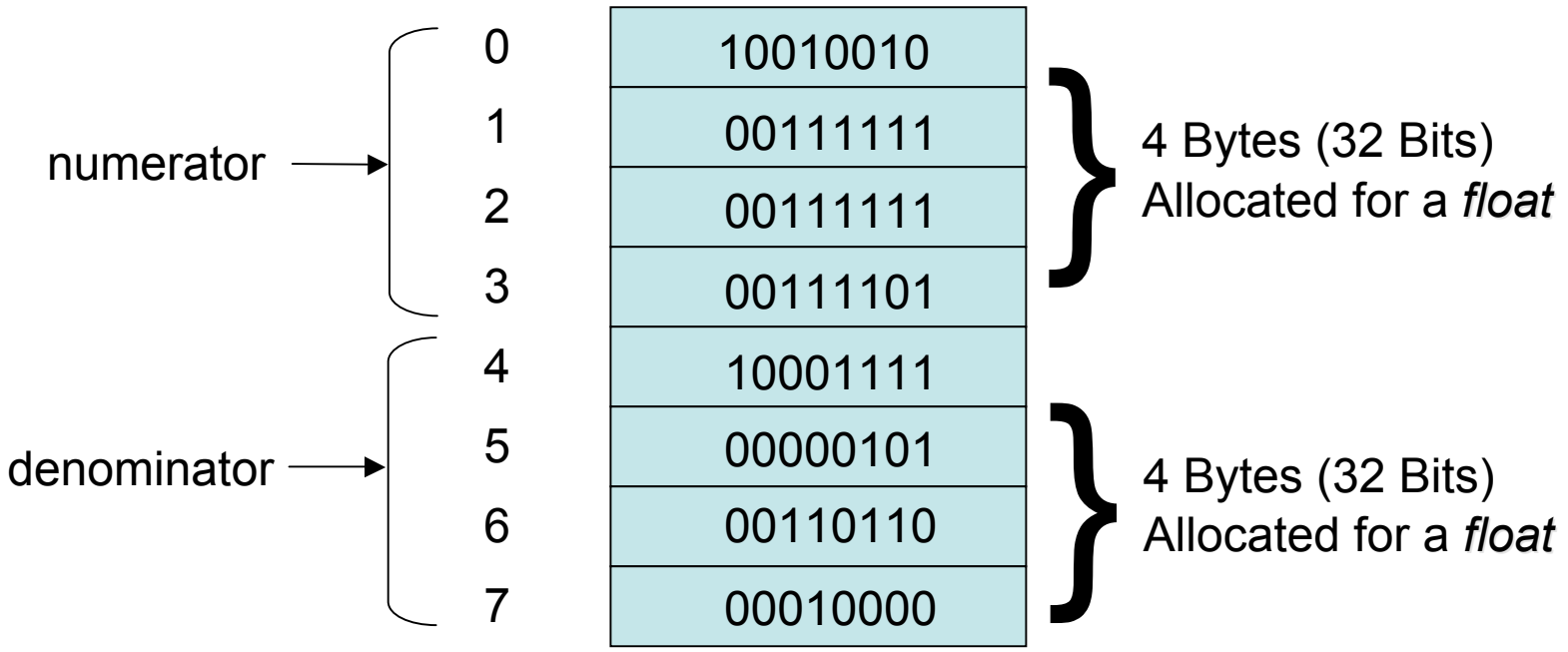
- Java naming convention dictates that variable names begin with lowercase, and then the first letter of successive words are capitalized. For example:

  int theFirstNum;

- This naming convention is known as “CamelBack” notation, since the capital letters mimic the humps on a camel's back.

Variables and Memory Maps

- Memory is laid out as a series of contiguous bytes (8-bits).
- By using variables, we have names that we can use for the storage locations of our values.



Variables and Memory Maps

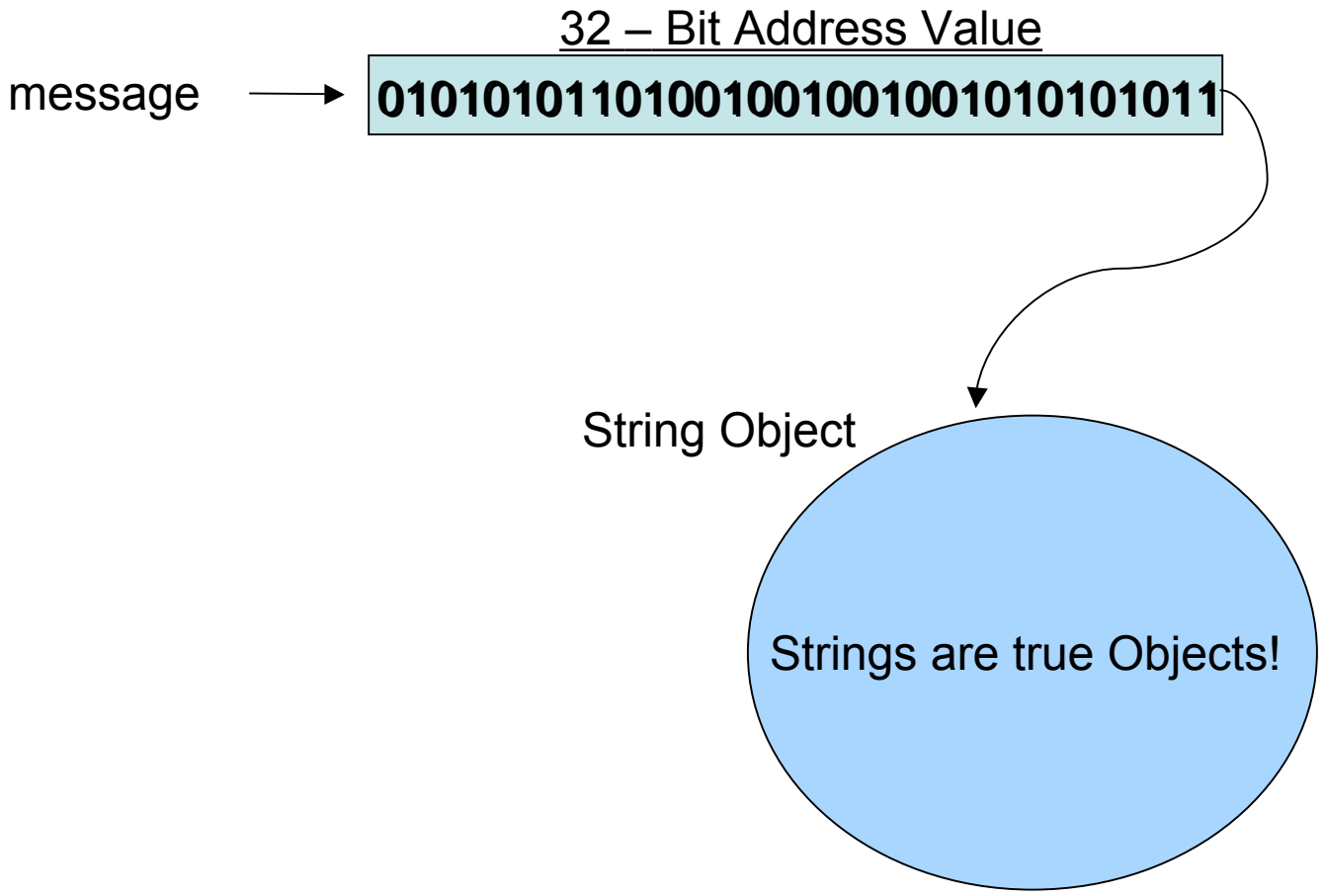
- *Objects* are dealt with a little differently.
- Variables that serve as names for *Objects* are called *reference variables*, since they refer to the *Object* that they name.
- Reference variables hold values that point or refer to the position in memory where the *Object* exists.
- String message = “Strings are true Objects!”;

  ...declares a *reference variable* named *message* that refers to a *String Object*.
- System.out.println(message);

  ...displays *Strings are true Objects!*.

Variables and Memory Maps

- The memory map view of how reference variables work looks something like this:



The reference variable message has a reference value (address) that refers to a String Object

Variables and Memory Maps

- The Power of Naming
  - The power we attain through the naming of variables is but one example of a more *generalized power of naming*.
  - We can name variables, methods, classes, objects, packages, and programs—each one of these serves as an abstraction. *Naming is abstraction.*
  - Much of Computer Science can be seen as the shrewd development and utilization of abstractions.
  - *Abstractions help us manage complexity.*

Class and Object Methods

- Method
  - A *method* is a *way of doing something*.
  - In Java, *a method is of set of one or more statements that are executed in succession, are given a name, and are invoked by that name*.
  - A method is analogous to a mathematical function, in that you often (not always) pass an argument into the function, the function performs some transformative process, and then a resultant value is returned from that function.
  - There are *two major categories of methods*, *Class methods and Object methods*.
  - *Class methods belong to the Class in which they are defined.*
  - *Object methods belong to each Object instantiated from a given Class.*

Class and Object Methods

- Class Methods
  - The general syntax for invoking a Class method is:

    ClassName.methodName();
  - Java programming convention dictates that Class names always begin with a capital letter, and method names always begin with a lowercase letter.
  - *Character* is an example of a *wrapper class*, in that the Objects created from it *wrap* a variable of the char primitive.
  - In other words, *a Character Object is an objectified char*.
  - To invoke a Class method, you do not need to have an Object instantiated from that Class (remember, Class methods belong to the Class, not to the Objects created from that Class).
  - System.out.println(Character.getNumericValue('Z'));

    ...displays *35*, which is the numeric value of the char value *Z*.

Class and Object Methods

- Object Methods
  - The general syntax for invoking a Object method is:

    objectReference.methodName();
  - Java programming convention dictates that Object reference names (i.e. reference variables) always begin with a lowercase letter.
  - String message = “Strings are true Objects!”;

    ...declares a reference variable named *message* that refers to a *String* Object.
  - String lowerMessage = message.toLowerCase();

    ...invokes the Object method named *toLowerCase()*, which creates and returns a new String Object that is the same as the original but with all lowercase characters. The returned value is then assigned to the reference variable *lowerMessage*.
  - System.out.println(message);

    ...displays *Strings are true Objects!*.
  - System.out.println(lowerMessage);

    ...displays *strings are true objects!*.

Class and Object Methods

- Object Methods (continued)
  - Note that, like all variables, *reference variables can be reused*.
  - System.out.println(message);

    ...displays *Strings are true Objects!*.
  - message = “Java rocks!”;

    ...assigns a reference value to *message* so that it points to a new String Object.
  - System.out.println(message);

    ...displays *Java rocks!*.
  - Also note that *multiple reference variables can refer to the same Object*!
  - String currentMessage = message;

    ...declares a reference variable named currentMessage which is then assigned a reference value that refers to the same *String* Object that *message* refers to.
  - System.out.println(currentMessage);

    ...displays *Java rocks!*.

The CLASSPATH

- The CLASSPATH is a delimited set of pathnames that the Java compiler searches to find the classes that it needs to load.
- Assuming you have installed the CD included at the back of our textbook, you will need to set your CLASSPATH so that you can access and use the classes that come with the CD.
- The CLASSPATH is easily set within DrJava by selecting...

  Edit->Preferences->Resource Locations

  and then click on the *Add* button.

  Locate the *PH_GUZDIAL/intro-prog-java/bookClasses* folder

  and click on the *Select* button.

Defining Classes and Creating Objects

- Our textbook, *Introduction to Computing & Programming with Java*, provides a set of media classes that facillitate learning Java and makes it more fun.
- World, Turtle, FileChooser, Picture, and Sound are just a few of these classes.
- You can find the Java source code for these classes in the PH_GUZDIAL/intro-prog-java/bookClasses directory.
- The source code files that contain the class definitions are Turtle.java, FileChooser.java, Picture.java, and Sound.java, respectively.
- As we progress through the textbook, these slides will probe some of the inner workings of the media classes.

Defining Classes and Creating Objects

- The general syntax for creating an object is:

  new ClassName(parameterList)
- The parameterList is a comma delimited list of parameters.
- If no parameters need to be passed in, then you do not need to provide a parameterList.
- System.out.println(new World());

  ...displays *A 640 by 480 world with 0 turtles in it.*

  ...additionally, *a Java JFrame is launched*.
- We can effectively do the same thing as in the above, only differently and in two steps:

  World world = new World(); // Creates a World, assigns to world.

  System.out.println(world);
- The advantage to this approach is that we still have our *world* reference that allows us to continue to use to our *World* object.

Defining Classes and Creating Objects

- Turtle fred = new Turtle(world);

  ...passes the *world* reference to the *Turtle* class constructor method.

  ...a *Turtle* object is instantiated (created), and its reference value is assigned to a *Turtle* reference variable named *fred*.

  ...you should now see your *fred* Turtle in the center of your *world*.

- Turtle george = new Turtle(20, 40, world);

  ...passes the same world reference to a Turtle class constructor that expects x and y coordinates in addition to a World reference.

  ...a *Turtle* object is instantiated (created), and its reference value is assigned to a *Turtle* reference variable named *george*.

  ...you should now see your *george* Turtle at x,y coordinates 20,40 in your *world*.

Sending Messages to Objects

- Now that we have *Turtle* reference variables named *fred* and *george*, we can "talk" to them by *sending them messages*.
- *Messages are sent to objects by invoking their methods.*

```
fred.setName("Frederick the Great");
george.setName("King George");
fred.turnRight();
fred.forward(50);
fred.turnRight();
fred.forward(50);
fred.turnRight();
fred.forward(50);
fred.turnRight();
fred.forward(50);
System.out.println(fred.getname());
```

Sending Messages to Objects

- That was a lot of messages to send to fred, just to draw a square. Wouldn't it be nice to be able to send just one message to fred, telling him to draw a square?
- By following the directions in our textbook, you can use the DrJava IDE to locate the Turtle.java source code file and add a drawSquare() method.
- Note that you will need to ensure that you have write access to the folders and files of the media source code (.java) files.
- The general syntax of a method is:

```
visibility type methodName( parameterList )
{
  // executable statements are placed here
}
```

Sending Messages to Objects

- Once you have added the drawSquare() method to the Turtle.java file (the Turtle class definition), you should be able to do the following:

```
World world = new World();
Turtle fred = new Turtle(world);
fred.drawSquare();
```

- Our textbook shows you how to write a drawSquare(int width) method that accepts a width argument.
- As an exercise, you should write a drawRectangle(int width, int length) method that accepts width and length arguments.

FileChooser and Picture

- *FileChooser* is a media class that launches a dialog box enabling you to browse your local file system to locate and select a file.
- *Picture* is a media class that accepts a *String* as parameter, where that String indicates the full pathname of a "*.jpg*" file.
- These classes can be used as follows:

```
String fileName = FileChooser.pickAFile();
Picture picture = new Picture(fileName);
picture.show();
```

Playing a Sound

- Similarly, sounds (.wav files) can be played as follows:

```
String fileName = FileChooser.pickAFile();
Sound sound = new Sound(fileName);
sound.play();
```

Object References and Primitives

- As alluded to previously, *object references are different from primitives*.
- *Primitives* (char, byte, int, short, long, float, double, and boolean) only have data associated with them—*primitives do not have methods*.
- *Object References* are variables that refer to true objects, and as such can be used to invoke methods upon (send messages to) the objects they refer to.
- *Primitive* example

```
int x;
x = 5;
```

- *Object Reference* example

```
String courseName = "Java II";
courseName.toUpperCase();
```

Backslashes and Slashes

- In Java, *backslashes* are used with String literals to denote what is called an *escape sequence*. Special characters are designated as an escape sequence. Here are some examples:

  \t tab

  \n newline

  \r carriage return

  \¨ double quote

  \\ backslash

- Note that \\ is the way that you indicate a backslash within a String literal.
- Alternatively, if you are trying to indicate a backslash character *as a delimiter within a filepath*, then you can *use the forward slash instead*.
- The following two examples are equilvalent, thought the second one is simpler and preferred:

  String photoName = "C:\\Pictures\\Vacation\\Mazatlan\\sunset.jpg";

  String betterName = "C:/Pictures/Vacation/Mazatlan/sunset.jpg";

- Using the slash character in this way is *platform independent* (it works for Windows, OSX, and Linux).