COP 3330, Spring 2013

Basic Java
(including intro to Objects, Classes)

Instructor : Arup Ghosh
01-11-13

School of Electrical Engineering and Computer Science
University of Central Florida

Java Comments

- Comments are designed to enhance the readability of source code.
- There are three styles of comments in Java:
- Line comments begin with // and consist of a single line only.
- Block comments begin with /* and end with */ and can cover many lines of commenting. Convention also puts an * in the leftmost position of every line in the comment.
- Javadoc comments begin with /** and end with */. They are used for documenting classes, data, and methods and can be extracted into an XHTML file using the JDK javadoc command. We’ ll deal much more with this type of comment later.

Reserved Words in Java

- Reserved words or keywords, are words that have a specific meaning to the compiler and cannot be used for any other purposes in a Java program.
- Note that Java is a case-sensitive language, which means that while `public` is a reserved word `Public` is not. However, from a readability perspective, it is best to avoid a reserved word in any form except that for which it was intended. (Note: `goto` and `const` are C++ reserved words not presently used in Java.)

Reserved Words in Java

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Modifiers in Java

- Java uses certain reserved words called modifiers that specify the properties of the data, methods, and classes and how they can be used.

Modifier	Applicable to					Explanation
	Class	Constructor	Method	Data	Block	
`(default)`	yes	yes	yes	yes	yes	A class, constructor, method , or data field is visible in this package. Default has no access modifier keyword.
`public`	yes	yes	yes	yes	no	A class, constructor, method , or data field is visible to all the programs in any package.
`private`	no	yes	yes	yes	no	A constructor, method, or data field is only visible in this class.
`protected`	no	yes	yes	yes	no	A constructor, method, or data field is visible in this package and in subclasses of this class in any package.
`static`	no	no	yes	yes	yes	Define a class method, or a class data field, or a static initialization block.
`final`	yes	no	yes	yes	no	A final class cannot be extended. A final method cannot be modified in a subclass. A final data field is a constant.
`abstract`	yes	no	yes	no	no	An abstract class must be extended. An abstract method must be implemented in a concrete subclass.

Identifiers in Java

- Identifiers are used in Java (as in other programming languages) to name programming entities such as variables, constants, methods, class, and packages.
- The rules for naming identifiers in Java are:
- An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word
- An identifier cannot be the words `true`, `false`, or `null`.
- An identifier can be of any length.
- Java is case-sensitive, so `X` and `x` are different identifiers.

Identifier Conventions in Java

- While identifier names should be as descriptive as possible, there are other style/convention guidelines that good programmers will follow to enhance the readability and maintainability of their code.
- The naming conventions for naming variables, methods, and classes are:
- Use lowercase letters for variables and methods. If a name consists of several words, concatenate them into one word, making the first word lowercase and capitalizing the first letter of each subsequent word. For example, `radius`, `getName`, `showInputDialog`.
- Capitalize the first letter of each word in a class name. For example, `ComputeArea`, `JOptionPane`, `ThisIsANewClass`.
- Capitalize every letter in a constant, and use underscores between words. For example, PI, MAX_VALUE.

Constants

- While the value of a variable may change during the execution of a program, the value of a constant cannot change (thats why its called a constant!).
- A constant must be declared and initialized in the same statement. A constant is defined in Java by using the keyword `final`.
- The syntax for a constant definition is:

```
final datatype CONSTANT_NAME = value;
```

- Java convention capitalizes every letter in a constant.

Numeric Operations

Java Operator	Meaning	Example	Result
+	Addition	34 + 1	35
–	Subtraction	34.0 – 1.0	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder (modulo division)	20 % 3	2

Modulo division can be quite useful. For example, any even number % 2 is always 0, and any odd number % 2 is always 1. So this is a simple way to determine if a number is odd or even. Suppose that today is Saturday, you and your friend are going to meet in 10 days. What day is in 10 days?

Saturday is the 6th day of the week

(6 + 10) % 7 = 16 % 7 = 2, thus you will meet on a Tuesday.

You will meet in 10 days. There are 7 days in a week Tuesday is the 2nd day of the week

Shorthand Operations

Java Operator	Meaning	Example	Result
`+=`	Addition assignment	x += 8	x = x + 8
`-=`	Subtraction assignment	x –= 4.0	x = x – 4.0
`*=`	Multiplication assignment	x *= 2	x = x * 2
`/=`	Division assignment	x /= b	x = x / b
`%=`	Remainder assignment	x %= 5	x = x % 5

Java Operator	Meaning	Description
`++var`	preincrement	`var` is incremented by 1, then the new value of `var` is returned.
`var++`	postincrement	`var` is returned (old value) then incremented by 1.
`--var`	predecrement	`var` is decremented by 1, then the new value of `var` is returned.
`var--`	postdecrement	`var` is returned (old value) then decremented by 1.

Numeric Type Conversions

- Sometimes it is necessary to mix numeric values of different types in a computation.
- Java automatically converts numeric types in an expression according to the following rules:
  1. If one of the operands is `double`, the other is converted into a `double`.
  2. Otherwise, if one of the operands is a `float`, the other is converted into a `float`.
  3. Otherwise, if one of the operands is `long`, the other is converted into a `long`.
  4. Otherwise, both operands are converted into an `int`.

Numeric Type Conversions

- You can always assign a value to a numeric variable whose type supports a wider range of values . This is called a widening conversion or widening a type. For example, you can assign a long value to a float variable. Java performs widening conversions implicitly.
- In Java. you cannot assign a value to a variable of a type with a smaller range of values (a narrowing conversion or narrowing a type) unless you use explicit type casting.
- Casting is an operation that converts a value of one data type into a value of another data type.

Numeric Type Conversions

- The syntax for casting is to place the target type in parentheses, followed by the variable or the value to be cast.

```
float f = (float) 10.1;
int I = (int) f;
```

- Casting does not change the variable being cast.

Character Type

- The increment and decrement operators also apply to variables of the `char` type.

```
char ch = ‘a’ ;
System.out.println(++ch);  //prints character b
```

- The `char` type only represents one character. To represent a string of character, use the data type called `String`. String is actually a predefined class in the Java library, just like the System class.
- The String type is not a primitive type, it is a reference type (an object).
- We will see `String class later.`

Casting Between `char` and Numeric Types

- A `char` can be cast into any numeric type and vice versa.
- When an integer is cast into a `char`, only its lower sixteen bits of data are used, the other part is simply ignored.

```
char c = (char)0XAB0041;
    //the lower 16 bit hex code 41 is assigned to c
  System.out.println(c);  //c is the character A
```

Casting Between `char` and Numeric Types

- When an floating-point value is cast into a `char`, the integral part of the floating-point value is cast into a `char`.

```
char t = (char)65.25;
    //decimal 65 is assigned to t
  System.out.println(t);  //t is the character A
```

Objects, Classes..

Abstractions

- Abstractions simplify complicated things.
- Instead of dealing with twenty low-level details, you can use maybe five high-level concepts instead.
- Military organization is an example of a powerful abstraction.
  - Try using a horde with no real chain of command and see how far you get.

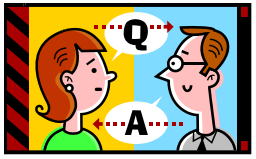
Programming languages

- Every language is an answer to the question, “What is the best set of abstractions for programming?”
- C uses the abstraction of separating variables (data) and functions (actions), and using the functions to transform the variables as required.
- Java uses the abstraction of *objects*, which are basically variables that know how to perform various actions.
- Objects are a Java abstraction to more closely model elements of a problem

Comparison

- Finding the length of a string in C:
  - int len = strlen(s);
  - Basically: “strlen(), take this string and give me its length.”

- Finding the length of a string in Java:
  - int len = s.length();
  - Basically: “You there, string s. How long are you?”



- A C string is just data. A Java string also has *behavior* – it can *do things* like figuring out its own length.

Objects

- Ultimately, an object is an abstraction for some element of a problem.

- For example, a chat program might have objects for:
  - Network connection
  - Friends (an object for each one)
  - Displaying the UI
  - And many more

- Java programs are just a bunch of objects passing messages to each other, asking for various actions to be performed.

Objects for a Chat Program

- networkConnection: “Friend bob has just come online. displayManager, put bob on the online friends list.”
- bob: “displayManager, I have a new avatar.”
- displayManager: “networkConnection, send bob’s new avatar.”
- networkConnection: “Retrieving…Here it is.”
- displayManager: “Displayed.”
- inputManager: “User just wrote a Message to bob. networkConnection, send it over. displayManager, move it from the input text area to the chat window.”

What it really looks like

- displayManager.addToOnlineFriends(bob);
- displayManager.notify(bob, NEW_AVATAR);
- Image avatar = networkConnection.requestImage(bob, CURRENT_AVATAR);

- And so on…

- This is a *very* rough approximation, and these lines of code come from three different places in the program.

Objects

- displayManager.addToOnlineFriends(bob);
- displayManager.notify(bob, NEW_AVATAR);
- Image avatar = networkConnection.requestImage(bob, CURRENT_AVATAR);

- These are *objects*: Basically just variables.

- A closer analog would be a C `struct`.
  - Objects have internal *state* : variables that live inside of them, as in a `struct`.

Methods

- displayManager.addToOnlineFriends(bob);
- displayManager.notify(bob, NEW_AVATAR);
- Image avatar = networkConnection.requestImage(bob, CURRENT_AVATAR);

- These are *methods*: Functions associated with an object.
  - Like normal functions, they take zero or more parameters.

- In the `struct` analogy, these are like functions that live inside the `struct`.

Another example: Method

- A method encapsulates an action or a service that an object of the class can perform when requested.

```
public class Person
  {
     private String name;
     public Person (String who)
     {
       this.name = who;
     }

     public String getName()
     {
       return name;
     }

  }
```

```
//create two Person objects
Person aGirl = new Person("Debi");
Person anotherGirl = new Person("Eva");
String girl1 = aGirl.getName();
//girl1 now has value of "Debi"
String girl2 = anotherGirl.getName();
//girl2 now has value of "Eva"
```

The `main` Method in Java

- Every Java application must have a user-declared `main` method where the program execution begins. (Note: Java applets do not have a main method.)
- The `main` method is always a `public static void` method.
- The `main` method has the following form (either one works):

```
public static void main (String[] args)
{
   //statements;
}
```

```
public static void main (String args[])
{
   //statements;
}
```

Classes

- displayManager.addToOnlineFriends(bob);
- displayManager.notify(bob, NEW_AVATAR);
- Image avatar = networkConnection.requestImage(bob, CURRENT_AVATAR);

- Here `Image` is a *class* – the *type* of the object `avatar`.

- Just like `5` is of type `int`, and `'x'` is of type `char`, `avatar` is of type `Image`.

- `displayManager` might be of type `UIHandler`, `networkConnection` could be a `NetworkInputStream`, etc.

MyFirstProgramInJava.. Remember?

- When we wrote `MyFirstProgramInJava`, we were creating a class.

- A new data type whose sole purpose is to greet the UCF.

- In Java, we create classes (or use previously defined ones) to represent aspects of the problem we're solving.
  - By making *objects* of a class, we can easily deal with these pieces.
  - The *methods* of the class allow these pieces to interact and perform appropriate actions.

Differences

Object

- An actual variable
- Has internal state
- Has behavior

Class

- A type: a *blueprint* for how to make an object
- Defines the structure of an object's state
- Defines the behavior that all objects of its type have

One class, Many objects

- Since a class is just a type, a single class can have multiple objects.
- For example, if we have a class Dog, it may have objects fido, rover, lassie, etc.
- The objects will have similar behavior (they are all Dogs), but their internal *state* may vary.
  - They have different names
  - Fido and Rover are male, Lassie is female
  - And so on…

Some reading material

- Getting started: http://docs.oracle.com/javase/tutorial/getStarted/index.html

- Basic OOP concepts: http://docs.oracle.com/javase/tutorial/java/concepts/index.html
  - Read the sections on objects and classes – you can skip the rest for now.

Summary

- Java programs are a collection of interacting objects
- A class is a blueprint (data type) for building an object