CMPSCI 187 Lecture #2

Java Overview

9 September 2011

Java Overview

- The Von Neumann Machine Model

- Primitive Data in Java

- Objects and Pointers

- Arrays and Strings

- Methods and Scope

The Von Neumann Machine Model

- Most computers have a **word size**, **home registers**, other **registers**, and an **instruction set**.

- These days 64-bit words can be numbers (fixed or floating point), characters, **pointers**, or instructions.

- An instruction normally either **moves** a word to or from a register or carries out some **operation** on a word in a home register.

- The program is a sequence of instructions, with control flow by **branch** and **goto** statements.

High-Level Languages, Compilers, Interpreters

- High-level languages like Java let us write programs without knowing the details of our machine, and give us a richer vocabulary of instructions.

- A Java program (one or more classes) is **compiled** into class files.

- When you run the program, an **interpreter** creates the machine-language program that the machine actually runs.

- We give up some efficiency and some care in memory management by going to a higher-level language, but we make up for it in programming power.

Primitive Data

- All data in Java eventually reduces to **primitive** values, usually stored in single machine words.

- There are several fixed-point primitive types (**byte**, **short**, **int**, **long**), two floating-point types (**float** and **double**), the Unicode character type **char**, and the **boolean** type.

- There are lots of arithmetic operations on these that you may look up.

- Some automatic **type casts** occur from some of these to others.

- Each primitive type has a **wrapper class** to allow them to be used as objects.

Objects and Pointers

- The basic unit of data in Java is an **object**, a collection of data with associated methods.

- Objects come from **classes**, and the **class definition** indicates what sort of data the object has and includes the code for its methods.

- Each object is stored somewhere out in memory, in a place chosen at run time. A **pointer** or **reference** is the address where an object is, stored in a machine word.

- While primitives are passed as parameters **by value**, objects are passed by **reference**. It is possible for two or more variables to refer to the same object, so that changes to one variable will affect the other -- this is called **aliasing**.

Dynamic Typing

- Look up "Haddock's Eyes" in Wikipedia -- "the name of the song is called..."

- An object has a **type** (what it is called) and a **class** (what it is).

- It gets a class when it is created with a **new** statement, and this never changes as the object is used.

- It can be referred to by a variable of any compatible type, e.g., a Dog variable could contain a Rottweiler or a Terrier object.

- When an **overloaded instance method** is called from an object, the version for the **class** is what gets run.

Arrays

• An array is a structure made up of primitives or objects of the same type.

• If T is any type, T[ ] is the type of arrays made from T objects.

• We can have two-dimensional arrays, or worse, with types like T [ ] [ ] [ ].

• Arrays of primitives are in consecutive locations -- arrays of objects are arrays of pointers, pointer to created objects (*picture*).

• Arrays are created like objects with **new** and have a given **length**. A method need not know the length of an array to use it.

• Shallow versus deep copying -- a new array with the same objects is aliased.

Strings

- Strings are more or less arrays of **char** but are a class of objects with many useful methods (see the API for reference).

- Strings are immutable -- once created they can only be overwritten with new strings, not edited. (Though if this happens within a method it looks like editing.)

- Remember that "=", as for objects, means "the same copy of the same String", while ".equals" means "the same characters in the same order".

- The "+" operator on Strings is concatenation, sometimes casting other things into Strings.

- All objects have "toString" methods.

Methods and Scope

- The commands that cause Java programs to actually do things all occur within **methods**.

- A class may have a **main method**, which can be run by the operating system.

- A main method may call other methods, which are either instance or class methods. A method call runs until a **return statement**, which resumes the calling method at the point after the call.

- Methods may have **return values** (of the type given in their signature) and/or **side effects**.

- Side effects could include changes in fields of objects, or creation of new objects.

Scope of Variables

- A field of an object is normally only modified by a method of that object's class (often called by some other method).

- A class may have **class variables**, also normally only modified by methods of the class.

- A variable declared within a method is meaningful only there, not outside and not within another method called from there.

- A method may have **parameters**, with names and types given by the method's **signature**.

- Methods and fields may be designated **public** or **private**.