Topic 1: The Java Programming Language

Part 1: Basics of Java

Java: First Class (“Hello World”)

Understanding the Statements that Produces the Output

- **Literal string**
  - Will appear in output exactly as entered
  - Written between double quotation marks
- **Arguments**
  - Pieces of information passed to a method
- Method
  - Requires information to perform its task
- `System` class
  - Refers to the standard output device for a system

Understanding the `First` Class

- Everything used within a Java program must be part of a class
- Define a Java class using any name or **identifier**
- Requirements for identifiers
  - Must begin with one of the following:
    - Letter of the English alphabet
    - Non-English letter (such as α or π)
    - Underscore
    - Dollar sign
  - Cannot begin with a digit

Understanding the `First` Class (cont'd.)

- Requirements for identifiers (cont'd.)
  - Can only contain:
    - Letters
    - Digits
    - Underscores
    - Dollar signs
  - Cannot be a Java reserved keyword
  - Cannot be `true`, `false`, or `null`
- **Access specifier**
  - Defines how a class can be accessed

Naming Conventions

- Packages: all-lowercase (more on this later)
- Classes: nouns, first letter of each internal word capitalized, DemoProgram
- Methods: verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized, getValue()
- Variables: mixed case, short and meaningful, stuNum for student number
- Constants: all uppercase, with words separated by underscores, MAX_WIDTH

http://www.oracle.com/technetwork/java/codeconventions-135099.html

the `main()` Method

- **`static`**
  - A reserved keyword
  - Means the method is accessible and usable even though no objects of the class exist
- **`void`**
  - Use in the `main()` method header
  - Does not indicate the `main()` method is empty
  - Indicates the `main()` method does not return a value when called
  - Does not mean that `main()` doesn't produce output

Method Calls and Placement

- **Method**
  - A program module
  - Contains a series of statements
  - Carries out a task
- Execute a method
  - **Invoke** or **call** from another method
- **Calling method** (**client method**)
  - Makes a **method call**
- **Called method**
  - Invoked by a calling method
- `main()` method executes automatically
- Other methods are called as needed

Access Specifiers

- Can be `public`, `private`, `protected` , or `package`
- `public` access allows use by any other class
- Also called access modifiers
- Methods most commonly use `public` access

(1-1)

Analyzing a Java Application that Produces Console Output

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

(Method.java)
(Dialog.java)

Adding Comments to a Java Class

- Types of Java comments
  - **Line comments**
    - Start with two forward slashes (//)
    - Continue to the end of the current line
    - Do not require an ending symbol
  - **Block comments**
    - Start with a forward slash and an asterisk (/*)
    - End with an asterisk and a forward slash (*/)

Adding Comments to a Java Class (cont'd)

- Types of Java comments
  - **Javadoc** comments (TriangleHelper.java)
    - A special case of block comments
    - Begin with a slash and two asterisks (/**)
    - End with an asterisk and a forward slash (*/)
    - Use to generate documentation

Java Data

Declaring and Using Constants and Variables

- **Constant**
  - Cannot be changed while program is running
- **Literal constant**
  - Value taken literally at each use
- **Numeric constant**
  - Constant associated with numeric datatypes
- **Unnamed constant**
  - No identifier is associated with it

Declaring and Using Constants and Variables

- **Variable**
  - A named memory location
  - Used to store a value
  - Can hold only one value at a time
  - Its value can change
- **Data type**
  - A type of data that can be stored
  - Determines how much memory an item occupies
  - Determines what types of operations can be performed on data
- **Primitive type**
  - A simple data type
- **Reference types**
  - More complex data types

Declaring and Using Constants and Variables

Keyword	**Description**
`byte`	Byte-length integer
`short`	Short integer
`int`	Integer
`long`	Long integer
`float`	Single-precision floating point
`double`	Double-precision floating point
`char`	A single character
`boolean`	A Boolean value (`true` or `false`)

Integer Data Types

Type	Minimum Value	Maximum Value	Size in Bytes
`byte`	–128	127	1
`short`	–32,768	32,767	2
`int`	–2,147,483,648	2,147,483,647	4
`long`	–9,223,372,036,854,775,808	9,223,372,036,854,775,807	8

Type	Minimum	Maximum	Size in Bytes
`float`	$-3.4 * 10^{38}$	$3.4 * 10^{38}$	4
`double`	$-1.7 * 10^{308}$	$1.7 * 10^{308}$	8

Using the `char` Data Type

- **`char`** data type
  - Holds any single character
- Place constant character values within single quotation marks

```
char myMiddleInitial = 'M';
```

- **`String`**
  - A built-in class
  - Stores and manipulates character strings
  - `String` constants are written between double quotation marks

(Datatype.java)

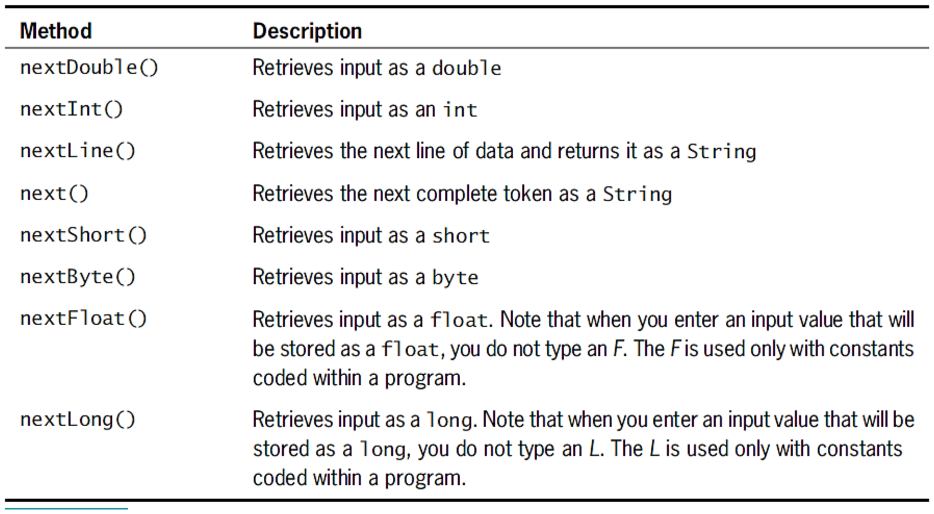
Using the `char` Data Type (cont'd.)

Escape Sequence	Description
\b	Backspace; moves the cursor one space to the left
\t	Tab; moves the cursor to the next tab stop
\n	Newline or linefeed; moves the cursor to the beginning of the next line
\r	Carriage return; moves the cursor to the beginning of the current line
\"	Double quotation mark; displays a double quotation mark
\'	Single quotation mark; displays a single quotation mark
\\\\	Backslash; displays a backslash character

Using the `Scanner` Class to Accept Keyboard Input

- `System.in` object
  - **Standard input device**
  - Normally the keyboard
  - Access using the `Scanner` class
- `Scanner` object
  - Breaks input into units called **tokens**

Using the `Scanner` Class to Accept Keyboard Input (cont'd.)



Method	Description
`nextDouble()`	Retrieves input as a `double`
`nextInt()`	Retrieves input as an `int`
`nextLine()`	Retrieves the next line of data and returns it as a `String`
`next()`	Retrieves the next complete token as a `String`
`nextShort()`	Retrieves input as a `short`
`nextByte()`	Retrieves input as a `byte`
`nextFloat()`	Retrieves input as a `float`. Note that when you enter an input value that will be stored as a `float`, you do not type an *F*. The *F* is used only with constants coded within a program.
`nextLong()`	Retrieves input as a `long`. Note that when you enter an input value that will be stored as a `long`, you do not type an *L*. The *L* is used only with constants coded within a program.

(Input.java)

Type Conversions

- Convert `String` to `int` or `double`
  - Use methods from the built-in Java classes `Integer` and `Double`
- **Type-wrapper classes**
  - Each primitive type has a corresponding class contained in the `java.lang` package
  - Include methods to process primitive type values

```
Integer.parseInt()
Double.parseDouble()
```

Understanding Type Conversion

- Arithmetic with variables or constants of the same type
  - The result of arithmetic retains the same type
- Arithmetic operations with operands of unlike types
  - Java chooses the unifying type for the result
- **Unifying type**
  - The type to which all operands in an expression are converted for compatibility
- Order for establishing unifying types between two variables (highest to lowest):
  1. `double`
  2. `float`
  3. `long`
  4. `int`

Explicit Type Conversions

- **Type casting**
  - Forces a value of one data type to be used as a value of another data type
- **Cast operator**
  - Place desired result type in parentheses
  - Using a cast operator is an **explicit conversion**
  - Does not change the type of the variable
- You do not need to perform a cast when assigning a value to a higher unifying type
  - Type Promotion
  - Type Demotion

Java Classes and Objects

Static

- Static method
    - Can be called on the class (no object required).
    - Also called class methods
- Static field
    - Shared by all instances of the class.
    - Also called class data
    - Often used for constants:
      private static final int NUM_STUDENTS = 42;
- Static class
    - A class that is useful to only one other class (embedded inside another class)

About Classes and Objects

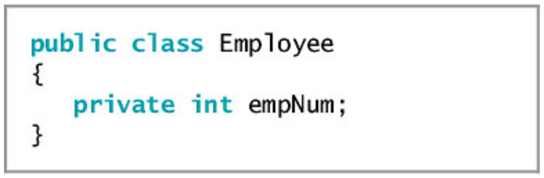
- Every object is a member of a class
- **Is-a relationships**
  - An object “is a” concrete example of the class
  - The zoo’s shark “is a” `Fish`
- **Instantiation**
  - Shark is an instantiation of the `Fish` class
- Reusability

About Classes and Objects (cont’d.)

- Methods are often called upon to return a piece of information to the source of the request
- **Class client** or **class user**
  - An application or a class that instantiates objects of another prewritten class

Creating a Class

- Assign a name to the class
- Determine what data and methods will be part of the class
- Create a class header with three parts:
  - An optional access modifier
  - The keyword `class`
  - Any legal identifier for the name of the class
- `public` class
  - Accessible by all objects



```
public class Employee
{
   private int empNum;
}
```

Creating a Class (cont'd.)

- **Extended**
  - To be used as a basis for any other class
- **Data fields**
  - Variables declared within a class but outside of any method
- **Instance variables**
  - Nonstatic fields given to each object
- **Private access** for fields
  - No other classes can access the field's values
  - Only methods of the same class are allowed to use `private` variables
- **Information hiding**
- Most class methods are `public`

Creating Instance Methods in a Class

- Classes contain methods
  - **Mutator methods**
    - Set or change field values
  - **Accessor methods**
    - Retrieve values
  - **Nonstatic methods**
    - **Instance methods**
    - “Belong” to objects
- Typically declare nonstatic data fields

Creating Instance Methods in a Class (cont’d.)

```
public class Employee
{
   private int empNum;
   public int getEmpNum()
   {
      return empNum;
   }
   public void setEmpNum(int emp)
   {
      empNum = emp;
   }
}
```

(Static.java)
(Static2.java)

Declaring Objects and Using Their Methods

- **Reference to the object**
  - The name for a memory address where the object is held
- **Constructor** method, i.e., Employee()
  - A method that creates and initializes class objects
  - You can write your own constructor methods
  - Java writes a constructor when you don't write one
  - The name of the constructor is always the same as the name of the class whose objects it constructs
- After instantiating an object, we can use the methods associated with the class.

Data Hiding

- Data hiding using encapsulation
  - Data fields are usually `private`
  - The client application accesses them only through `public` interfaces
- `set` method
  - Controls the data values used to set a variable
- `get` method
  - Controls how a value is retrieved

Using Constructors

`Employee chauffeur = new Employee();`

- Actually a calling method named `Employee()`

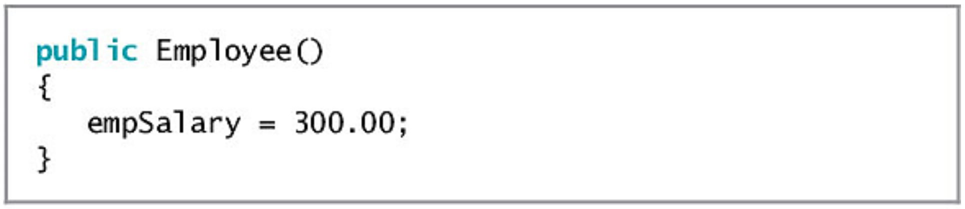
- **Default constructors**
  - Require no arguments
  - Created automatically by a Java compiler
    - For any class
    - Whenever you do not write a constructor

Using Constructors (cont'd.)

- The default constructor provides specific initial values to an object's data fields
  - Numeric fields
    - Set to 0 (zero)
  - Character fields
    - Set to Unicode '\u0000'
  - Boolean fields
    - Set to `false`
  - Nonprimitive object fields
    - Set to `null`

Using Constructors (cont'd.)

- A constructor method:
  - Must have the same name as the class it constructs
  - Cannot have a return type
  - `public` access modifier



```
public Employee()
{
    empSalary = 300.00;
}
```

Understanding That Classes Are Data Types

- Classes you create become data types
  - Often referred to as **abstract data types** (**ADT**s)
    - Implementation is hidden and accessed through public methods
  - **Programmer-defined data type**
    - Not built into the language
- Declare an object from one of your classes
  - Provide the type and identifier

(Rectangle.java)