Introduction to Java

CS 3: Computer Programming in Java
Objectives

- Begin with primitive data types
- Create a main class with helper methods
- Learn how to call built-in class methods and instance methods
Primitive Data Types

- Rules for programmer created identifiers
- Examples of valid and invalid programmer created identifiers
- Multiple declarations
- Java strings
Rules for Programmer Created Identifiers

- Program will need to create memory or storage locations for storing the program data and method names as well as choose names for classes and packages
- The job of the programmer is to create these identifiers
The rules for the programmer created identifiers in Java are as follows:

- Identifiers must only contain alphabets, underscore ( _ ), digits (0 to 9 ), and a $ sign.
- An identifier must begin with an alphabet, underscore or $.
  - It cannot begin with a digit.
- An identifier cannot contain blank spaces.
  - A horrid practice used by Windows filing system.
- An identifier cannot be a Java reserved word.
Examples of Valid Identifiers

- A
- a1
- student_name
- stdntNm
- _aSystemName
- _anthrSysNm
- $class
- _public
Examples of Invalid Identifiers

- Sum+var
  - The ‘+’ sign is illegal; violates the first rule
- 2names
  - The first character is invalid; violates the second rule
- Stdnt Nmbr
  - Contains whitespace in between the phrases; violates the third rule
- int
  - Reserved word; violates the fourth rule
More About Identifiers

- It is important to use meaningful identifiers
- Another Java coding practice is to start the name of classes with upper case letter
- If the name of the class contains more than one word, then first letter of each word is capitalized
The names of methods begin with lower case letter, but if method name contains more than one word then first letter of each word after the first word is capitalized.

Names of program and class variables may be in upper or lower case, but it is a good programming practice to use all lower case names for Java primitive data types and first letter of each word capitalized for the instances of Java classes or objects.

```java
int myint = 10;
float mydecimal = 10.0006;
```

Leaving out data type in Java for a local or class variable name in which data will be stored will cause a compile error.
Types of Primitive Data Types

- Integral
  - byte
  - char
  - short
  - int
  - long
- Boolean
  - Can only have values true and false
- Floating Point
  - float
  - double
# Java Primitive Sizes and Values

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (bytes)</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>2</td>
<td>Uses Unicode characters; language dependent</td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
<td>-9223372036854775808</td>
<td>9223372036854775807</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>+/-1.4E-45</td>
<td>+/-3.4028235E+38</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>+/-4.9E-324</td>
<td>+/-1.7976931348623157E+308</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Multiple Declarations

- Java allows one to use the comma operator and declare multiple variables of the same data type on one line.
- For example, the following declarations are legal:
  - `int val1, val2, val3;`
  - `int val4=22, val5=-10, val6;`
- However, understand that a compile error will result if any attempt is made to use `val1`, `val2`, `val3`, and `val6` prior to their initialization.
Strings in Java are objects of class java.lang.String

- This means that String class is located in the java.lang package
- Also, this package is automatically imported in all Java programs
- Java String class is designed in such a manner that two or more String objects and Strings and primitive Java data can be concatenated, simply by using the plus (+) sign
Java String Concatenation Example

For example, the following String concatenation is a legal Java construct:

```java
String Big_String = "John" + ' ' + 'H' + '.' + ' ' + "Smith is " + 20 + 
" years old and has a gpa = " + 3.9 + " and he does well in " + true + "/" + 
false + " type examinations.";
System.out.println (Big_String);
```

The above code snippet will print the following on the DOS console:

John H. Smith is 20 years old and has a gpa = 3.9 and he does well in true/false type examinations.
Notice that using the property of overloaded operator plus (+) for String class, Java is able to concatenate characters (such as space, alphabets, and punctuation marks), integers (20 in above case), floating-point numbers (3.9), and Boolean literals (true and false) all into one string.

It is however, required that on right hand side of String concatenation expression, at least one of the data must be of String type.
Invalid Java String Concatenation Example

- For example, expression given below is not a valid String concatenation expression and will cause compile error:
  
  String Not_A_String = 'H' + 25 + 'L'; // Compile error. None of the expressions to be concatenated are of type String!

- Therefore, one must remember the following String concatenation rule: At least one of the expressions to be concatenated must be of String type
Create a Main Class with Helper Functions

- Some Programming projects are simple enough to be solved by a single main method which uses built-in Java Classes and their predefined methods.
- However, most robust applications will need to be modularized, both within the Main class and within the user defined Classes that will be needed to implement them.
Simple Java Application:
Example #1

- The following is an example of a simple application which solves a simple problem of reading a name and an age from the user, and displaying the number of months the person has lived
- The first version solves the problem using only the main method
Simple Java Application:
Example #2

- Although this approach will solve simple programs, we want to divide the program into modules which become methods that perform simple tasks to support the overall goals of the application.

- When these methods are placed in the Main class to assist the main method, they are called Helper Methods.

- The following listing shows a better version of the previous application, using helper methods.

```java
import java.util.Scanner;
import java.text.DecimalFormat;

public class InCalcOut2 {
    // InCalcOut2.java Author: Edwin Ambrosio
    // Program calculates how many months old you are after
    // prompting you for your name and age.
    // Demonstrates the use of the Helper methods in the Main
    // class of InCalcOut1.java
    public static void main (String[] args) {
        String name;
        int age;
        double months;

        name = getName();
        age = getAge();

        // Code to calculate months from age and display results
    }
}
```
```java
Simple Java Application:  Example #2 (2)

months = age * (double)365 / 30;

// Round the output to three decimal places
DecimalFormat fmt = new DecimalFormat("0.###");

System.out.println(name + " you are " + fmt.format(months) + " months old.");
}

// Reads a character string from the user and returns it to main

public static String getName()
{
    String name;
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter your name:");
    name = scan.nextLine();
    return name;
}
```
Simple Java Application: Example #2 (3)

// -----------------------------------------------
//  Reads an int from the user and returns it to main
// -----------------------------------------------

public static int getAge()
{
    int age;
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter your age: ");
    age = scan.nextInt();
    return age;
}
Why Method 2?

- In this second application, we have used helper methods getName(), and getAge() to assist perform the duties of obtaining data from the user.
- We expect this type of design/structure in all the programs that you write for this course.
- Do not try to get by with writing a large main method to solve the problem.
Built-In Class Methods

- All methods are either Class Methods or Instance Methods
- Class methods are defined within a Class using the static modifier in its description
  - This allows the method to be called by using the Class name as the qualifier: `Classname.methodname`
In Java there is no operator to perform the exponent operation, so, we must call the `pow` method from the Math Class.

The code to square the variable `radius` and put its value into the variable `rsquared` (assuming all variables are type double) would be:

```java
rsquared = Math.pow(radius, 2);
```

When `main` calls `getName()` and `getAvg()`, these calls are to Class methods, but do not need to be qualified because the calls occur within the Class definition.
**Instance Methods**

- Do not have the static modifier in their definition, and must be called by using a Class object reference as the qualifier.
- Are intended to manipulate the data contained in the particular instance/object that called it.
- All the remaining calls to methods in the above examples are calls to Instance Methods.
Instance Methods Example

- The call:
  
  ```java
  name = scan.nextLine();
  ```
  
  - calls the `nextLine()` method of the Scanner Class for the instance object `scan`

- The call:
  
  ```java
  System.out.println("Enter your name: ");
  ```
  
  - calls the `println()` method of the System Class for the instance object `out`.  

Summary

- **Identifiers**
  - Choose names for classes and packages
  - **Rules**
    - Identifiers must only contain alphabets, underscore ( _ ), digits (0 to 9 ), and a $ sign
    - An identifier must begin with an alphabet, underscore or $.
    - An identifier cannot contain blank spaces.
    - An identifier cannot be a Java reserved word.

- **Primitive Data Types**
  - Integral
  - Boolean
  - Floating point
Summary

- Java strings
  - Strings in Java are objects of class `java.lang.String`
  - On the right hand side of String concatenation expression, at least one of the data must be of String type

- Creating a Main class with helper functions
  - Most robust applications will need to be modularized, both within the Main class and within the user defined Classes that will be needed to implement them

- Built-in methods and instance methods
  - Class methods are defined within a Class using the static modifier in its description
  - Instance methods do not have the static modifier in their definition, and must be called by using a Class object reference as the qualifier