Java Programming 2 – Lecture #2 – Jeremy.Singer@glasgow.ac.uk

Java Primitive Types

Java has eight *primitive* types defined and supported in the language and underlying virtual machine. These are shown in the table below.

Type	Description	Default initial value
`byte`	8-bit signed integer value	`0`
`short`	16-bit signed integer value	`0`
`int`	32-bit signed integer value	`0`
`long`	64-bit signed integer value	`0L`
`float`	32-bit single precision IEEE 754 floating point value	`+0.0F`
`double`	64-bit double precision IEEE 754 floating point value	`+0.0`
`boolean`	boolean value	`false`
`char`	16-bit Unicode character value	`\u0000`

Since Java is a statically typed language, types must be declared for:

- local variables e.g. `int i;`
- class fields e.g. `class C { boolean b;}`
- method return and parameter values e.g. `public float getValue(long l) {…}`

Matching Values with Types

Match the following values with their corresponding primitive types from the table above.

```
false    1.3e-9    'a'    -256  256   1L   Double.POSITIVE_INFINITY
```

Java Identifiers

An *identifier* is the textual label for a named Java entity, such as a class, field, method, parameter, local variable… There are rules and conventions for identifiers.

The main rule is that an identifier must begin with a letter or an underscore, and must be at least one character long. Non-initial characters may be letters, numbers or underscores. Note that since Java supports the Unicode character set, letters are not restricted to the 26 characters of the Latin alphabet.

Conventions are not enforced by the Java compiler, but should be observed by careful programmers to make source code easier to understand. Coding standards and guidelines will specify different conventions. Near-universal conventions include:

- initial capital letter for class name, initial lower case for other identifiers
- multi-word identifiers are encoded in CamelCase e.g. `ArrayIndexOutOfBoundsException`
- constant values have all caps identifiers e.g. `Math.PI`

Simple For Loops

The Java `for` loop has the same semantics as in C. The three clauses in the parentheses are for initialization, termination condition check and post-loop-body update of the iteration variable(s). Thus:

```
for (int i=0; i<10; i++) { doSomething();}
```

is equivalent to:

```
int i; while (i<10) { doSomething(); i++;}
```

Below is a simple method that counts the number of vowel characters in a `String` object.

```
/**
 * count the number of vowel chars [aeiou] in a String
 * @arg s String to process
 * @return number of vowel chars in s
 */
public static int numVowels(String s) {
  int vowels = 0;
  for (int i=0; i<s.length(); i++) {
    char c = s.charAt(i);
    if (c=='a' ||
        c=='e' || …) {
      vowels++;
    }
  }
  return vowels;
}
```

Notice the use of `String` API (application programmer interface) methods `length()` and `charAt()`. We can investigate the full set of `String` methods via the online Java API[1].

Questions

1) Check out the `switch` control-flow construct for Java. Can you replace the above `if` statement to check for vowels with a switch?

2) Is `void` a primitive type in Java?

[1] Just google for **Java String API Oracle** and the appropriate webpage should be top of the search results.