Java Basics

Ye Yang
Stevens Institute of Technology

Understanding Memory

- The smallest measurable unit of memory is 1 bit, holding a zero(0) or one(1)
- A byte is typically defined as smallest storage necessary to hold one ASCII character
- One byte is 8 bits and 256 (i.e. 2^8) different characters can be represented using one byte
- ASCII table has 128 characters (i.e. 0-127)
  - American Standard Code for Information Interchange

ASCII Characters

- Some examples of ASCII characters are

  ‘A’ = 65

  ‘B’ = 66

  ….

  ‘ ‘ = 32

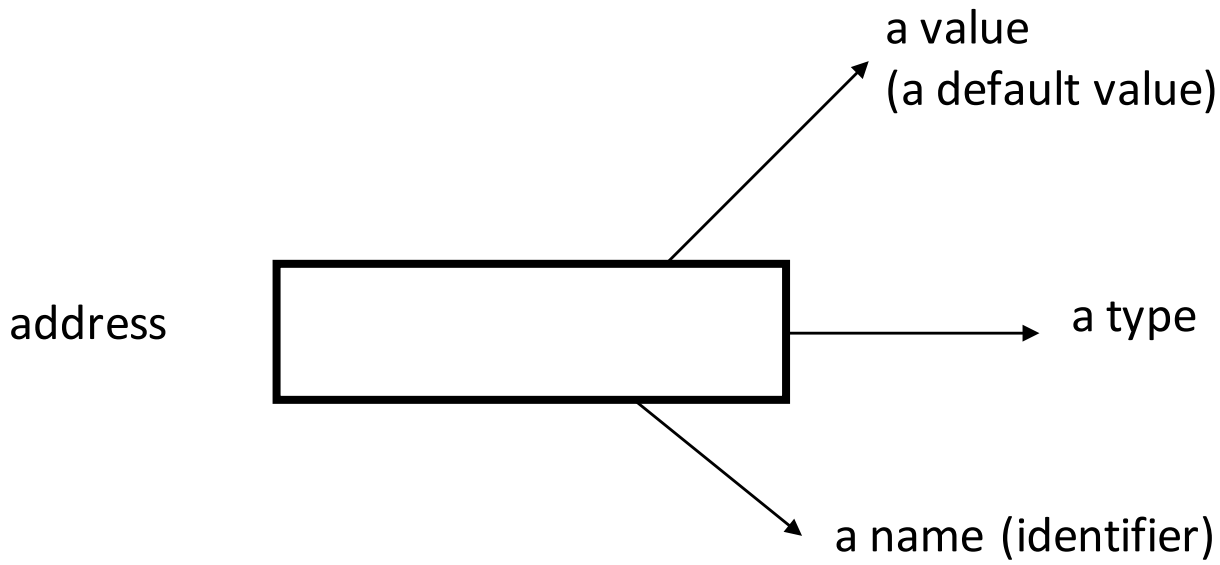
  ‘a’ = 97

- Express them in binary

Variables and Types

- A variable is a location in memory where values can be stored and referenced.
- Variables are associated with types, which have particular data sizes and a restricted set of valid operations that can be performed on them.
- Variable declarations assign an identifier, a type, and may declare an initial value. Many types have a default value. For example, an uninitialized integer defaults to 0.

Variable (a memory location)



The address or value may be passed as a parameter

Identifiers

- Identifiers are used in Java to give a name to classes, methods and variables that can be used to access them. Java identifiers begin with a "*letter*," followed by letters or digits, and, in a small number of cases, underscores ("_").
- There is a Java standard naming convention for the use of identifiers, which can be found at
  - http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
- Google Java Style Guide
  - https://google.github.io/styleguide/javaguide.html

Sample Code Standards

- Here are a few examples of code standards from the section on declarations:
  - 6.1 One declaration per line is recommended since it encourages commenting. In other words,

```
int level; // indentation level
int size; // size of table
```

is preferred over

```
int level, size;
```

  - 6.2 Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces “{” and “}”.)
  - 6.3 Try to initialize local variables where they’re declared.
  - …
- There are a lot of details to learn, and it will take a while to be proficient in Java coding.

Types in Java

- 8 Primitive Types
  - boolean, byte, char, short, int, long, float, double
- Reference Types, a.k.a. non-primitive types or user defined types
  - used to reference object, which stores the memory address where the object resides
    - Default value of null
    - Used to invoke an object's methods
  - strings, arrays, and file streams, etc.
- Important differences between these two.

Primitive Types

Primitive Type	What it Stores	Range
byte	An 8-bit (1-byte) integer value	-128 to 127
short	A 16-bit (2-byte) integer value	-32,768 to 32,767
int	A 32-bit (4-byte) integer value	-2^{31} to $2^{31}-1$
long	A 64-bit (8-byte) integer value	-2^{63} to $2^{63}-1$
float	A 32-bit (4-byte) floating-point value	6 significant digits (10^{-46}, 10^{38})
double	A 64-bit (8-byte) floating-point value	15 significant digits (10^{-324}, 10^{308})
char	A 16-bit (2-byte) character using the Unicode encoding scheme	
boolean	A boolean value (size dependent on JVM)	True and false

Constants

- Integer constants
  - In either decimal, octal, or hexadecimal notation
    - Octal: indicated by a leading 0;
    - Hexadecimal notation: indicated by a leading 0x or 0X;
    - For example, representing integer 37:
      - Decimal: 37; Octal: 045; Hexadecimal: 0X25
- Character constant
  - enclosed by a pair of single quotation marks, as in ‘a’
- String constant
  - Enclosed within double quotation marks, as in “Hello”
- Special sequences (escape sequences)
  - ‘\n’, ‘\\’, ‘\’’, and ’\”’ mean, respectively, the newline character, backslash character, single quotation mark, and double quotation mark

Declaration and Initialization of Primitive Types

- Examples:
  - int num3;
  - double minimumWage = 4.50;
  - int x = 0, num1 = 0;
  - int num2 = num1;

- Here are a few examples of code standards from the section on declarations:
  - 6.1 Use one declaration per line to encourage comments
  - 6.2 Try to initialize local variables where they are declared
  - 6.3 Put declarations only at the beginning of blocks
- There are a lot of details to learn, and it will take a while to be proficient in Java coding.

Assignment Operator (=)

lvalue = rvalue;

```
w = 10;
x = w;
z = (x - 2)/(2 + 2);
```

- Take the value of the rvalue and store it in the lvalue.
- The rvalue is any constant, variable or expression.
- The lvalue is named variable.

Mathematical Operators

- Operations on int, long, short, and byte types.
  - Addition +
  - Subtraction -
  - Multiplication *
  - Division /
    - rounds toward zero (drops the remainder)
  - Modulus %
    - calculates the remainder of x / y

- Except for "%", these operations are also available for doubles and floats.
- Floating-point division ("/") doesn't round to an integer, but it does round off after a certain number of bits determined by the storage space.

Simple Arithmetic

```
public class Example {
    public static void main(String[] args) {
      int j, k, p, q, r, s, t;
      j = 5;
      k = 2;
      p = j + k;
      q = j - k;
      r = j * k;
      s = j / k;
      t = j % k;
      System.out.println("p = " + p);
      System.out.println("q = " + q);
      System.out.println("r = " + r);
      System.out.println("s = " + s);
      System.out.println("t = " + t);
    }
}
```

```
> java Example
p = 7
q = 3
r = 10
s = 2
t = 1
>
```

Shorthand Operators:
+=, -=, *=, /=, %=

Common	Shorthand
a = a + b;	a += b;
a = a - b;	a -= b;
a = a * b;	a *= b;
a = a / b;	a /= b;
a = a % b;	a %= b;

Shorthand Operators

```
public class Example {
    public static void main(String[] args) {
      int j, p, q, r, s, t;
      j = 5;
      p = 1; q = 2; r = 3; s = 4; t = 5;
      p += j;
      q -= j;
      r *= j;
      s /= j;
      t %= j;
      System.out.println("p = " + p);
      System.out.println("q = " + q);
      System.out.println("r = " + r);
      System.out.println("s = " + s);
      System.out.println("t = " + t);
    }
}
```

```
> java Example
p = 6
q = -3
r = 15
s = 0
t = 0
>
```

Shorthand Increment and Decrement `++` and `--`

Common	Shorthand
a = a + 1;	a++; or ++a;
a = a - 1;	a--; or --a;

Increment and Decrement

```
public class Example {
    public static void main(String[] args) {
      int j, p, q, r, s;
      j = 5;
      p = ++j;  //  j = j + 1;  p = j;
      System.out.println("p = " + p);
      q = j++;  //  q = j;     j = j + 1;
      System.out.println("q = " + q);
      System.out.println("j = " + j);
      r = --j;  //  j = j -1;   r = j;
      System.out.println("r = " + r);
      s = j--;  //  s = j;     j = j - 1;
      System.out.println("s = " + s);
    }
}
```

```
> java example
p = 6
q = 6
j = 7
r = 6
s = 6
>
```

java.lang library

- The java.lang library has more operations in...
  - the Math class.
    - x = Math.abs(y); // Absolute value. Also see Math.sqrt, Math.sin, etc.
  - the Integer class.
    - int x = Integer.parseInt("1984"); // Convert a string to a number.
  - the Double class.
    - double d = Double.parseDouble("3.14");

Converting types

- Integers can be assigned to variables of longer types.

```
int i = 43;
long l = 43; // Okay, because longs are a superset of ints.
l = i;       // Okay, because longs are a superset of ints.
i = l;       // Compiler “type mismatch” ERROR.
i = (int) l;   // Okay.
```

- The string "(int)" is called a cast, and it casts the long into an int.

Boolean Values

- A boolean value is either "true" or "false".
- Logical operators (for Booleans):
  - "&&" (and), "||" (or), and "!" (not)

a	b	a && b	a || b	!a
false	false	false	false	true
false	true	false	true	
true	false	false	true	false
true	true	true	true	

Boolean Variables

- Boolean values can be specified directly ("true", "false")
- Or created by the comparison operators "==", "<", ">", "<=", ">=", "!=" (not equal to).

```
boolean x = 3 == 5;       // x is now false.
x = 4.5 >= 4.5;           // x is now true.
x = 4 != 5 - 1;           // x is now false.
x = false == (3 == 0);    // x is now true.
```

Conditional Statements and Loops

- Relational and equality operators
- Logical operators
- The if statement
- The while statement
- The for statement
- The do statement
- Break and continue
- The switch statement
- The conditional operator

The if Statement

- The *if statement* has the following syntax:

The *condition* must be a boolean expression. It must evaluate to either true or false.

`if` is a Java reserved word

```
if ( condition )
    statement;
```

If the *condition* is true, the *statement* is executed. If it is false, the *statement* is skipped.

The if Statement

- An example of an `if` statement:

```
if (sum > MAX)
   delta = sum - MAX;
System.out.println ("The sum is " + sum);
```

- First the condition is evaluated -- the value of `sum` is either greater than the value of `MAX`, or it is not
- If the condition is true, the assignment statement is executed -- if it isn't true, it is skipped.
- Either way, the call to `println` is executed next

Indentation

- The statement controlled by the `if` statement is indented to indicate that relationship
- The use of a consistent indentation style makes a program easier to read and understand
- Although it makes no difference to the compiler, proper indentation is crucial to human readers

Nested if Statements

- The statement executed as a result of an `if` statement or an `else` clause can be another `if` statement
- These are called *nested if statements*
- An `else` clause is matched to the last unmatched `if` (no matter what the indentation implies)
- Braces can be used to specify the `if` statement to which an `else` clause belongs

- Some long chains of if-then-else clauses can be simplified by using a "switch" statement.

Repetition Statements

- *Repetition statements* allow us to execute a statement multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements:
  - the *while loop*
  - the *do loop*
  - the *for loop*
- The programmer should choose the right kind of loop for the situation

The while Statement

- A *while statement* has the following syntax:

```
while ( condition )
    statement;
```

- If the condition is true, the statement is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- The statement is executed repeatedly until the condition becomes false

The do Statement

- A *do statement* has the following syntax:

```
do
{
    statement;
}
while ( condition )
```

- The statement is executed once initially, and then the condition is evaluated
- The statement is executed repeatedly until the condition becomes false

The for Statement

- A *for statement* has the following syntax:

The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
    statement;
```

The *increment* portion is executed at the end of each iteration

The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

The for Statement

- An example of a `for` loop:

```
for (int i=1; i <= 100; i++)
   System.out.println (i);
```

- The initialization section can be used to declare a variable
- Like a while loop, the condition of a for loop is tested prior to executing the loop body
- Therefore, the body of a for loop will execute zero or more times

`for` loop Exercises

- How many times is the loop body repeated?
  - `for (int x = 3; x <= 15; x += 3)`
    `System.out.println(x);`
  - `for (int x = 1; x <= 5; x += 7)`
    `System.out.println(x);`
  - `for (int x = 12; x >= 2; x -= 3)`
    `System.out.println(x);`
- Write the `for` statement that print the following sequences of values.
  - 1, 2, 3, 4, 5, 6, 7
  - 3, 8, 13, 18, 23
  - 20, 14, 8, 2, -4, -10
  - 19, 27, 35, 43, 51

The “break”, “continue” and “return” keywords

- ”break” exits the innermost loop only (frequently used in conjunction with the switch statement).
- “continue” allows to only give up current iteration of a loop and go on to the next iteration.
- “return” causes a method to end immediately. The flow of control returns to the calling method.

```
switch (month) {
  case 2:
    days = 28;
    break;
  case 4:
  case 6:
  case 9:
  case 11:
    days = 30;
    break;
  default:
    days = 31;
    break;
  }
```

```
if (month == 2) {
  days = 28;
} else if ((month == 4) || (month == 6) ||
        (month == 9) || (month == 11)) {
  days = 30;
} else {
  days = 31;
}
```

```
public int daysInMonth(int month) {
  switch (month) {
  case 2:
    return 28;
  case 4:
  case 6:
  case 9:
  case 11:
    return 30;
  default:
    return 31;
  }
}
```

JAVA APIs

- Think Java API (Application Programming Interface) as a super dictionary of the Java language.
  - It has a list of all Java packages, classes, and interfaces; along with all of their methods, fields and constructors.
    - java.lang intrinsic classes (String, etc)
    - java.io reading and writing
    - java.util Java Collection Framework and utility classes
    - javax.swing GUI
- Think Java API as the interface to manipulate Java classes as black boxes
  - It tells you how to use Java classes but little how they are implemented

JAVA APIs (cont'd)

- Any serious Java programmers should use the APIs to develop Java programs
  - Best practices of using APIs
- APIs Specifications online:
  - http://www.oracle.com/technetwork/java/api-141528.html
  - Treat it as a dictionary for reference

Strings and String

- Strings are an inevitable part of any programming task.
  - Printing messages (e.g. instant messaging with friends, output debugging information)
  - Representing data (e.g. student names)
  - Referring to files on disk (e.g. “c:\\file.txt”)
- Intuitively, think strings as a sequence of character
- In Java, String has an array of char as its internal representation.