Core / Intermediate Java

Help Notes

January 2016, Simon Roberts

Basic File Structure

- Any `public` class must be in a file named matching the class name (including case).
  - Therefore limited to one `public` class per source file.
- Package name should be all lower case, dotted form must match directory structure.
- Everything must be in a class.
  - General code must be in a method.
  - Code not in a class causes very strange error messages.

Program Entry Point

Declare `main` method in a `public` class, as follows:

```
public static void main(String[] args)
  throws Throwable {
    <code>
}
```

- The `throws` clause is for *classroom convenience*, and should normally be avoided.
- The name `args` is arbitrary, but conventional.
- Other than the two notes above, this declaration must be exact!

Variable declaration

<type> <identifier> [= <literal or expression>] ;

Identifiers should start with a letter, combinations of letters and numbers may follow.

Essential Coding Conventions

- Class names are `InitialCapsCamelCase`
- Method and variable names are `camelCaseWithInitialLower`
- "Constants" are `ALL_CAPS_WITH_UNDERSCORE`
- K&R style braces (i.e. "`{`" goes at the *end* of the line)

Essential literal formats

`123`	`int` number
`1_234`	Grouping format for any number
`123L`	`long` number
`12.3`	`double` number
`12.3E+10`	`double` number
`12.3F`	`float` number
`'x'`	`char`
`"xyz"`	`String`
`'\n'` and others	good in `char` and `String`

`int [] an_array =` `  { 1, 3, 5, 7, };`	Array initialization literal (note *optional* trailing comma)
`doStuff(new int[]` `  { 1, 3, 5, 7 });`	Array literal for other than initialization

Essential Operators

`+`, `-`, `*`, `/`	Add, subtract, multiply, divide
`++`, `--`	Increment / decrement (prefix or postfix)
`+`	String concatenation
`%`	Modulus (also known as “remainder”)
`[` <int> `]`	Array subscript selection
`&&`, `||`, `!`	Logical and, or, and not (`&&`, `||` "short circuit")

Notes:

- Precedence is broadly normal; use parentheses if unsure!
- Many operators support "assignment operator" forms:
  `a += 20; // add 20 to a`
- Arithmetic produces at least `int` or the larger of mixed operand types.

Comparison Operators

`<`, `<=`, `>=`, `>`	Less, less or equal, greater or equal, greater
`==`	Equals
`!=`	Not equal

Notes:

- `==` tests equality of *variables*, which are likely to be references, *not* the objects those references point at.
- Use `x.equals(y)` for most object comparisons.
- Comparison operators form `boolean` expressions.

Ternary / Conditional Operators

Set `x` to `"it is!"` if `test` is true, otherwise `"it's not"`:

```
String x = test ? "it is!" : "it's not";
```

Notes:

- The type of a ternary expression is not limited to String, provided the two alternative expressions are compatible (e.g. both are `int` values.)
- The alternative values can be any valid expression.

Conditional Constructs

```
if ( <boolean expr> ) { <code> }
[ else { <code> } ]


// switch requires int, String, or enum
switch ( <control expression> ) {
case <constant expr1>:
  <code>
  break; // falls through without this!
case <constant expr2>:
case <constant expr3>:
// sequential cases allow "or" type behavior
...
[ default: ] // if no case matches
```

Iteration Constructs

```
while ( <boolean expr> ) { <code> }


do { <code> } while ( <boolean expr> );
```

C-style "for" loop

```
for ( <inits> ; <boolean expr> ;
      <increments> ) {
  <code>
}
```

<inits>	Variable declaration/initialization
<increments>	Expressions with side effects, e.g. increments

Loop over contents of array or other "bucket" type

```
for ( <type> <var> : <bucket of type> ) {
  <code>
}
```

e.g.

```
String[] names = {
  "Fred",
  "Jim",
  "Sheila"
};


for (String n : names) {
  System.out.println("> " + n);
}
```

Declaring A Method

```
[ Modifiers … ]
<return type> <identifier> ( [<arguments>] )
[throws <exception> [ , <exception> ]…] {
  <code>
}
```

Notes:

- Modifiers might be `public static`
- Return type is mandatory, use `void` if nothing is returned
- Method name must be legal identifier
- Argument list is optional, form is comma separated list of type+variablename pairs
- In classroom, the `throws` clause may simply be `throws Throwable` until discussed fully

Examples:

```
public static String makeMessage(
    String name, boolean isMale) {
  return "Greetings "
    + (isMale ? "Mr." : "Ms.") + name;
}
public void idle() { // no args or return
}
public static int doubleIfPositive(int v) {
  if (v >= 0) {
    return 2 * v;
  else {
    return v;
  }
}
```

Calling A Method

Examples:

Call a `static` method in the same class, with a single argument:

```
doStuff(“A message”);
```

Call a `static` method with no arguments, in class `OtherClass`:

```
OtherClass.doOtherStuff();
```

Call an instance method on a `String` object `myName`, store the result in a variable `myShoutedName`:

```
myShoutedName = myName.toUpperCase();
```

Notes:

- Calling a non-static method from a `static` method (such as `main`) *requires* an explicit instance variable prefix.
- Calling a non-static method from a non-static method in the same class will use an implied prefix `this` if none is provided explicitly.

Java API Documentation

Find it on Oracle's website.

Bookmark it, and consider downloading it.

Make using it a habit, it's more reliable and up-to-date than Google!

Console Output

Output followed by newline:

```
System.out.println("Text " + val + "more");
```

Output without a newline:

```
System.out.print("Enter text: “);
```

Fancy formatted output:

```
System.out.printf("%2$10s : %1$7.3f\n",
98.4, "Temp");
```

Notes:

`2$`	Optional, selects the position of the argument to substitute
`\n`	Inserts a newline, does not have to be at the end of the line

Keyboard input

Setup:

```
Scanner sc = new Scanner(System.in);
```

Read a line:

```
String line = sc.nextline();
```

Note:

- For reading multiple lines, see the **File Input** section next.

File Input

Note that most IDEs run programs with the "current working directory" set to the *root directory of the project*.

Setup:

```
Scanner sc = new Scanner(
  Files.newBufferedReader(
    Paths.get("blah.txt")));
```

Loop to read all the lines:

```
while (sc.hasNextLine()) {
  String line = sc.nextline();
  // process the text
}
```

Note:

- This loop can be used with keyboard input too, but EOF *might fail* in an IDE.

Generating Random Numbers

Generate `double x` such that `0 <= x < 1.0`

```
double x = Math.random();
```

Generate `int x` such that `a <= x < b`

```
int x = ThreadLocalRandom.current()
          .nextInt(a, b);
```

Convert `String` And Other Types

Any `Object` Type → `String`

```
myObject.toString()
```

Any Type → `String`

```
"" + value
```

`String` to number types

```
int i = Integer.valueOf("123");
double d = Double.valueOf("3.2");
```

`String` to `enum` value

```
DayOfWeek dow = DayOfWeek.valueOf(
    text.trim().toUpperCase());
```

`int` to `enum` value

```
DayOfWeek dow = DayOfWeek.values()[index];
```

`enum` value to `int`

```
int index = DayOfWeek.SATURDAY.ordinal();
```

Split a `String` Into Separate Words

Split based on “non-word” characters (spaces, commas, etc.)

```
String line = "Hello there, how are you?";
String [] words = line.split("\\W+");
```

Split based on specific characters (colon in this example)

```
String [] words = line.split(":");
```

Creating Generic Collections

A `Set` containing `String` objects:

```
Set<String> myStrings = new HashSet<>();
```

A `List` containing the same strings that are in another collection:

```
List<String> orderedStrings =
  new LinkedList<>(myStrings);
```

A `Map` that pairs `String` primary keys with `Integer` values:

```
Map<String, Integer> table =
  new HashMap<>();
```

Sorting a `List` (Java 8)

Assuming the contents of the list have a “natural order” (e.g. for `List<String>`:

```
myList.sort(null);
```

Note, this sort *modifies* the original list.

Sorting a `List` (Pre Java 8)

Assuming the contents of the list have a “natural order” (e.g. for `List<String>`:

```
Collections.sort(myList);
```

Note, this sort *modifies* the original list.

Get Keys Of A `Map<String, Integer>`

```
Set<String> keySet = myMap.keySet();
```

Get Today's Date (Java 8)

```
LocalDate ld = LocalDate.now();
int day = ld.getDayOfMonth();
int month = ld.getMonthValue();
int year = ld.getYear();
```

Get Today's Date (Pre Java 8)

```
Calendar c = Calendar.getInstance();
day = c.get(Calendar.DAY_OF_MONTH);
month = c.get(Calendar.MONTH) + 1;
year = c.get(Calendar.YEAR);
```