Java Review - DPR 104

Java

- Java is an object-oriented, professional programming language.
- A java program is made of building blocks called classes.
- Every class is a container for variable declarations and method declarations.

Class	Contents	Description
System	out	variable representing console output
	in	variable representing keyboard input
Math	pow()	static method for exponentiation
	sqrt()	static method for square root
Scanner	nextInt()	non-static method to read integer from keyboard
	nextDouble()	non-static method to read decimal from keyboard
String	length()	non-static method yielding length of string object
	substring()	non-static method yielding a piece of the string object

- To run a java program requires a Java Runtime Environment (JRE) which contains a Java Virtual Machine (JVM). The JVM turns the actual machine into a java machine.
- To compile (build) a java program requires a Java Development Kit (JDK) which contains a java compiler and a JRE. Usually programmers use an Integrated Development Environment (IDE) to write java programs. Eclipse is a popular, open source, IDE.

Data Types

- In order to understand the memory used by a program a data type is assigned to it. The data type determines the amount of storage and how to interpret the bits within the storage.
- Java is a statically typed language. Every piece of data needs a type.
- There are two kinds of java types: primitive and reference (object).

Primitive Types

- `boolean`: true / false.
- `char`: 16-bit Unicode character.
- `byte`: 8-bit signed arithmetic.
- `short`: 16-bit signed arithmetic.
- `int`: 32-bit signed arithmetic.
- `long`: 64-bit signed arithmetic.
- `float`: single precision scientific.
- `double`: double precision scientific.
- By default all integer computations use `int` unless an operand is `long`.
- There is no detection of integer overflow.
- By default all scientific computations use `double`.

Reference Types

- Everything that's not a primitive type is a reference type.
- `String` is a class, therefore `String` is a reference type.
- An array is a reference type.
- Implemented as a pointer to another area of memory.

Literals

Every primitive type has a literal form. String objects also have a literal form.

Type	Literal	Description
`boolean`	`true`	`false` is the other Boolean literal
`char`	`'A'`	Single quotation marks are delimiters
`int`	`123`	No decimal point
`double`	`123.0`	Must have decimal point
`String`	`"abc"`	Double quotation marks are delimiters

Variables

- A variable is a <u>name for a piece of memory</u>. A variable must be declared and initialized before it can be used.
- Instance variable - Data member of a class that is instantiated for each object of the class.
- Static variable - Data member of a class that is not instantiated for each object of the class and belongs to the class itself.
- Local variable - variable declared in a method and is instantiated for each invocation of the method.

Example:

```
class MyClass {
  String x;      // instance variable
  static int y; // static variable

  public void f() {
    double z;    // local variable
    . . .
  }
}
```

Java Comments

```
// starts an end of line comment.
/* starts in in-line comment.  Must be terminated by */
```

Java Review - DPR 104

Java Naming Conventions

Class name

- Starts with a capital letter
- Is a simple or compound noun spelled in camel-case (e.g. `InputStream`)
- File name must match class name w/ .java extension

Variable name

- Starts with a lower-case letter
- Is a simple or compound noun spelled in camel-case (e.g. `availableBalance`)

Method name

- Starts with a lower-case letter
- Is a simple or compound verb spelled in camel-case (e.g. `transferMoney`)

For loop indices

- Usually just `i`, `j`, `k`, etc. for historical reasons

Expressions

- Expressions are combinations of variables, literals, and method calls. The combinations must follow the syntax of the language but is modeled after arithmetic expressions, e.g. 1+2
- Some expressions compute numerical (`int` or `double`) values.
- Some expressions compute logical (`boolean`) values.
- The possible computations are determined by the operators of the language.

Operators

- Assignment: `=`
- Arithmetic: `*, /, %, +, -` (`%` is modulus operator)
- Arithmetic extended assignment operators: `*=, /=, %=, -=` (not important)
- String concatenation operator: `+`
- Variable increment and decrement operators: `++, --`
- Relational operators: `<, <=, >, >=, ==, !=` (== means equals, != means does not equal)
- Boolean logical operators: `!, &&, ||`
    - `&&` is short-circuit 'and' operator
    - `||` is short-circuit 'or' operator
- Operators have precedence and binding order (left to right / right to left). Use parentheses to control the order of evaluation.

Declaration Statement

You must declare a variable before you can use it. The syntax is:

```
<type> variableName;
```

A variable can be initialized where it is declared.

```
<type> variableName = <expression>;
```

Assignment Statement

- The assignment statement gives a value to a variable. The syntax is: `variable = expression;`
- The equals sign is the assignment operator. It does not mean "is equal to."
- The type of the expression must be compatible with the type of the variable. For shortening assignments a cast is required.

if Statement

- The if statement is fundamental for the expression of logic. The syntax is:

```
if (<logical expression>) {
  // true statement(s)
}
```

-or-

```
if (<logical expression>) {
  // true statement(s)
} else {
  // false statement(s)
}
```

- if statements can be nested. This nesting may reflect mutually exclusive categories:

```
if (x == 1) {
  // x's value is 1.
} else if (x == 2) {
  // x's value is 2.
} else if (x == 3) {
  // x's value is 3.
} else {
  // x's value is NOT 1, 2, or 3.
}
```

while Statement

- The while statement is fundamental for repetition. The syntax is:

```
while (<logical expression>) {
  // repeated statement(s)
}
```

- As long as the logical expression evaluates to 'true' the repeated statement is executed.

for Statement

- The for statement is also for repetition. The syntax is:

```
for (<initialization>; <logical expression>; <update>) {
  // repeated statement(s)
}
```

- Note that the separator is ; (semicolon) not , (comma).

- The above is equivalent to the following while statement:

```
<initialization>;
while (<logical expression>) {
  // repeated statement(s)
  <update>;
}
```

Methods

- A method (function) is a way to group statements so that they can be called many times from different parts of a program. It is even a good idea to group statements into a method when it is called once.
- A method (function) is a fundamental building block for any program.
- The syntax for a method declaration is:

```
public static <type> methodName( <formal parameter list> ) {
  // statement(s)
}
```

- Note that the parentheses following the method name are always required as are the matching pair of curly braces that follow.
- The formal parameter list is a comma-separated list of declarations with the syntax:

```
<type> parameterName
```

Parameter Passing and Transfer of Control

- The parameters (arguments) declared in the method declaration are called formal parameters. The values passed to the method during it's invocation are called actual parameters.
- When a method is invoked all of the expressions that correspond to actual parameters are evaluated so as to produce single values, then these values are assigned to the formal parameters. Then the method is "invoked" ("called").
- At the point of invocation, execution control is transferred to the method statements. These statements may themselves invoked other methods. This can lead to a long "call chain." When a method returns, execution continues at the point just past its invocation.

Return value / Return statement

- A method may return a value as per its `<type>`. If a method does not return a value its `<type>` must be `void`.
- The `return` statement is followed by an expression when the method returns a value. If the method does not return a value the `return` keyword is not followed by an expression.
- A void method can "fall off the end" in which case a `return` statement is not needed.

String Class

- A string literal is a series of characters enclosed in double quotation marks.
- `String` is also a class that contains methods for manipulating string objects.
- `+` is the one operator that works with strings. It means "concatenate" or "append to."
- A good way to think about a string is an array of characters.

```
literal          array/index
                                         1 1
                  0 1 2 3 4 5 6 7 8 9 0 1
"Hello World!"   |H|e|l|l|o| |W|o|r|l|d|!|
```

- With this analogy the following String methods make sense
  - `charAt(index)` - get the character at the index position
  - `length()` - # of characters in the string
  - `substring(start)` - get the substring starting at start index for rest of string
  - `substring(start, end)` - get the substring starting at start index up to end index - 1.

Comparing Strings

- Use the non-static `equals()` method to compare two strings (case-sensitive) for equality.
- Use `equalsIgnoreCase()` method to compare two strings (case-insensitive) for equality.
- To compare to strings lexicographically (alphabetically) use the `compareTo()` method.

Arrays

- An array is collection of similar values that can be referenced by a variable.
- An array variable must be declared, allocated space, and initialized before it can be used.
- Array declaration: `type [] variable;`
- Array space allocation: `variable = new type [size];`
- Array initialization usually involves a for loop.
- Arrays can be declared, allocated, and initialized in one step using an array initializer value.
  - Example: `int [] smallPrimes = {2, 3, 5, 7, 11, 17, 19};`
- An array has a built-in `length` property that yields the size of the current allocation. Example: `smallPrimes.length`.
- Array elements are referred to by their index (or subscript) value. Example: `smallPrimes[3]`
- Array indexing starts at zero. The last element of an array has index `length - 1`.
- At run-time every array index operation is checked. If the index is not in range a `ArrayIndexOutOfBoundsException` is generated.
- Processing an array generally requires a looping statement.
- An array variable is implemented as a reference (pointer) variable.

Arrays Class

- The `Arrays` class has useful static methods for manipulating arrays:
  - `copyOf()` - return a copy of the array

  - `fill()` - fill the array elements with a specific value
  - `sort()` - sort (ascending) the elements of the array
  - `toString()` - return a string representation of the entire array

Arrays as Parameters and Return Values

- When an array is passed as a method parameter only its reference (pointer) value is passed.
- When an array is returned as the value of a method, a new array must be allocated and initialized in the method. A reference (pointer) value is returned to the caller.

Common Array Algorithms

- These are common array algorithms. You should be able to quickly code each of these:
  - Filling
  - Sum and Average Value
  - Linear Search
  - Swapping Elements
  - Copying Arrays
  - Finding Maximum and Minimum Value

Miscellaneous Statements

- The switch statement acts like a series of nested if-else statements. It requires the use of the break statement within it.

```
switch (<expression>) {
case <number1>:   <statements> break;
case <number2>:   <statements> break;
...
default:   <statements>
}
```

- The `continue` statement is used within a loop statement. It "continues" the loop at the top on the next iteration.
- The `break` statement is used within a loop statement. It exits the loop immediately.
- The conditional operator is used in an expression to yield either the true or false expression.

```
<expression> ? <true expression> : <false expression>
```

Text Input

- Use a `Scanner` type object to read text from an `InputStream`. To read from the console (keyboard) use the paradigm below:

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

- The `Scanner` class has many methods designed to convert textual data to the Java representation for a type. Some Scanner methods are:
  - `nextInt()` - convert the next sequence of characters to an `int` value
  - `nextDouble()` - convert the next sequence of characters to an `double` value
  - `next()` - convert the next sequence of characters delimited by whitespace to a string
  - `nextLine()` - convert the remainder of the line to a string.

Text Output (Printing)

- The `OutputStream` class has the methods for printing:
  - `print()`
  - `println()`
  - `printf()`
  - These are accessed by the `System.out` variable. Example: `System.out.print().`
- `print()` always requires a parameter and prints its value on the current line.
- `println()` usually takes a string parameter and prints it value and then starts a new line.
- `printf()` usually takes two or more parameters. The first parameter is required. It is a formatting string (see `Formatter` class) which intermixes text with substitution placeholders. The remaining parameters provide the substitution values.
- Common formatting codes (substitution placeholders):
  - `%d` - integer
  - `%f` - floating point
  - `%c` - character
  - `%s` - string

Character Processing

- Many applications need to process characters from a string. The `Character` class has the following static methods to do this:
  - `isLetter()` - return true if the character is a letter of the alphabet
  - `isDigit()` - return true if the character is numeric
  - `isWhitespace()` – return true if the character is a whitespace character.
- Whitespace characters have the following literal representation:
  - `\n` - end of line (eol)
  - `\r` - carriage return
  - `\t` - tab
  - `\f` - form feed
  - ` ` - a blank space.
- The whitespace characters can be used within a string literal (e.g. "hello\n") or a character literal (e.g. '\n').