L32

Monday, January 31, 2022 6:06 PM

Fundamentals of java.lang class, Math Class, Autoboxing and Unboxing

java.lang Package

Q1. java.lang is a package which contains many fundamental classes used in writing Java code. It is present in JDK and JRE installations.

For example, it contains classes like :

1. Object - The Object class is the root class of all classes in Java.
2. System - The System class provides access to the standard input, output and error streams of the running Java program. It also has many other useful methods.
3. Wrapper Classes - Whenever we want to represent a primitive int value as an object, we use its wrapper class Integer. The java.lang package contains all the wrapper classes: Boolean, Byte, Character, Short, Integer, Long, Float and Double.
4. String, StringBuilder classes.
5. Math - The Math class contains many useful methods for numeric operations like the square root, power, trigonometric operations etc.
6. Class - The class called Class represents the loaded classes and interfaces in memory during runtime.
7. Other classes - The java.lang package also has many other useful classes like ClassLoader, Process, Runtime, Thread, Runnable, Throwable, Exception, Error, etc.

All public classes present in java.lang package (like the classes mentioned above) are by default imported in every Java class. Meaning, we can directly start using these classes without using the import statement.

- [x] The `System` class is present in `java.lang` package.
- [ ] The `Date` class is present in `java.lang` package.
- [x] The class called `Class` is present in `java.lang` package.
- [ ] We must declare `import java.lang.System;` statement for code statements like `System.out.println("Hello");` to work.

Math Class

Q1. The Math class present in java.lang package contains many useful static methods. These static methods are also called as utility methods.

Classes like Math that have only static methods are also called Utility classes.

Note: Most of the methods in Math class use double as data type, so we will see fractional part in the results.

```java
package q11239;
public class MathExamples {
    public static void main(String[] args) {

        double base = 4;
        double exponent = 3;
        System.out.println(base + " to the power " + exponent + " is : " + Math.pow(base, exponent));
        System.out.println("Square root of " + base + " is : " + Math.sqrt(base));

    }
}

```

Coding Conventions

Q1. Code should be written in such a way that fellow programmers can understand easily. Most of the time the code written by one author is maintained by someone else. In such a scenario code would be easy to read and maintain if all authors followed some common rules or conventions..

Java language designers have provided guidelines called Java code conventions for writing Java code.

Java naming rules for classes, interfaces, methods and variables employ camel-case convention. In camel-case convention, when two or more words are joined together the starting letter of each word joined will be written in uppercase.

For example: SimpleDateFormat, getAge, maxSpeed.

Below are the rules for naming:

1. **Class** - A class name should be a noun and not a verb. For example
   NumberPrinter is preferred over PrintNumbers. It should follow camel-case and the starting letter should also be in **uppercase**. Eg: String, StringBuilder, ArrayList
2. **Interface** - The same rules followed for a class name.
3. **Method** - A method name should be a verb and not a noun. For example
   getAge() is preferred over age(). It should follow camel-case and the starting letter should be in **lowercase**. Eg: clear(), toString(), resetURLConnection()
4. **Variable** - A variable name should be a meaningful and clearly indicate what it is going to store. It should be a noun. For example
   maxAge is preferred over x. It should follow camel-case and the starting letter of the first word should be in **lowercase**. Eg: speed, maxSpeed
   Single-letter variables like x, y, z - should never be used.
   Single-letter variables like i, j, k etc - can be used as loop-counters in for loops.
5. **Constant** - A constant variable is one which is declared as static and final. Its name should be a meaningful and clear like a simple variable, but it does not follow camel-case. It is written completely in uppercase letters with '_' as a word separator. Eg: public static final int MAX_WIDTH = 100;

According to Java naming conventions, select all the correct statements in the below sample code :

```
public class ParseText { // statement 1
public int GET_LENGTH(String textcontent) { // statement 2
return textcontent.length(); // statement 3}
}
```

- [ ] The class name `ParseText` is correct in Statement 1.
- [ ] In Statement 1, the class name does not follow camel-case convention. It should have been `parseText`.
- [ ] The method name `GET_LENGTH` in Statement 2 is correct.
- [x] In Statements 2 and 3, the variable name `textcontent` should have been written as `textContent`.

Q2. According to Java code conventions, a **blank line** is recommended to be used between two sections in a class. For example, between two methods.

Below are the rules for using a space ( ):

1. Between **keyword** which is followed by **parenthesis**:
   ```
   while (condition) {
           ...
   }
   for (..; ...; ...) {
           ...
   }
   ```
   Note : There **should not** be a space between method name and the parenthesis that follows it.
2. A space should appear after commas. For example:
   ```
   public int sum(int num1, int num2) {
           ...
   }
   String[] namesArr = {"Aries", "Taurus", "Gemini", "Cancer"};
   ```
3. A space should be used between all **binary operators** and their **operands** except dot (.). **Unary operators** (-, -- and ++) **must never be separated** from their operands by a space. For example:
   ```
   int total = num1 + num2;
   for (int i = 0; i < namesArr.length; i++) {
           ...
   }
   ```
4. A space should be used between expressions of a for statement. For example:
   ```
   for (int i = 0; i < namesArr.length; i++) {
           ...
   }
   ```

Select all the correct statements that follow the above rules in the below code:

```
public boolean isPrimeNumber (int number) { //statement 1
for(int i=2; i<=(number/2); i++) {//statement 2
if ((number%i)==0) { //statement 3
return false;
                }
        }
        return true;
}
```

- [x] In statement 1, there should not be a space between the method name and the opening parenthesis. isPrimeNumber (

  Yes, space should be given only between keywords and the opening parenthesis which follows them.

- [x] In statement 2, there should be a space between `for` and the opening parenthesis that follows it.
- [x] In statement 2, there should be a space on either side of `=` (assignment operator) in the initialization expression `int i=2;`.
- [x] In statement 2, there should be a space on either side of `/` and `<=` operators in the condition expression `i<=(number/2);`.
- [x] In statement 2, there should be a space after the semicolon `;` of the condition expression `i<=(number/2);`.
- [x] In statement 3, there should be a space on either side of `%` and `==` operators in the **if**'s condition expression `(number%i)==0`.