Language Fundamentals

9) Identify correctly constructed package declarations import statements class declarations (of all forms) including inner classes) interface declarations and implementations (for java.lang.Runnable or other interface described in the test) method declarations (including the main method that is used to start execution of a class) variable declarations and identifiers.

package, import, classes (including inner classes) :

Package is a collection of grouped classes. eg, java.lang, java.util

- The fully qualified name of a class consists of the package name and the class name, separated by "."
- Every class is a member of some package.
- Interpreter and compiler will look for a class such as Math.class in the directory \java\lang\
- The root of the search is a directory on the CLASSPATH.
- so to import all classes under the directory \java\lang\ your import statement should look like *import java.lang.*;*

Ordering of Java source file elements:

- When writing a package, the name of the package should be the first line in the source file.
- import statements if any should be the next one and it should be before the source code of the class.
- Next comes the public classes, followed by any non−public classes. Eventhough, the order of public and non−public classes doesn't matter usually the public class comes first.
- A source file can contain atmost one public class definition, and the file name must match the name of the public class.
- You can have as many top−level classes in one source file, but only one can be declared public and the source file name must match the public class name
- A source file can contain any number of non−public classes, but each will be compiled into a separate .class file.
- Having a public class in a source file is not a must. Even if a source file doesn't have a public class the compiler won't complain.
- Comments are the exception. It can come anywhere in a source file, even before package statement.

eg,

```
//This is just an example package
package myPack;
import java.lang.*;

/* main class
Source filename : Test.java
*/
public class Test {
.....
}
class myTest extends Test{
.....
}
class Test2 {
.....
}
```

Interface declarations and implementations :

- An interface is a collection of abstract method definitions and constants − no instance variables and no method implementations. variables must be declared public, static and final.
- Interfaces provides much of the functionality of multiple inheritance, but without the difficulties.
- Interface is a skeleton of a class. The class which implements it should define those methods including their bodies.
- The declarations in an interface are always public, even if you don't label them so.
- Any variables declared in an interface must be static and final−−that is, they must be constants, even if not labelled "final".
- An interface looks a lot like an abstract class, except that it uses the keyword interface instead of the words abstract and class
- While an abstract class may define some abstract methods and some non−abstract methods, all the methods defined within an interface are implicitly abstract.

creating an interface :

```
public interface test {
public void print();
}
```

*Note :*There is no curley braces after the print method.

This code promises that any class that implements the test interface will have a print method that will take a test object regardless of whether or not its superclass promises the same. All descendants of such a class would implement test.
To implement test in your class,

```
class A extends B implements test {
public void print() {
System.out.println("test");
}
}
```

- A class can *implement* any number of interfaces, whereas can *extend* only one parent class
- You can extend an interface
- interfaces are not instantiated with *new*, but you can declare as,

```
test t = new A();
```

Remember :

- An interface is a collection of abstract method definitions and constants − no instance variables and no method implementations.

Abstract Class	***Interface***
must not be instantiated	must not be instantiated
may contain static and final data	variables are implicitly static and final
abstract class can have	methods are implicitly abstract.

non−abstract methods but, abstract method should be inside an abstract class	Therefore, all methods should be implemented in the subclass which implements it. no method implementation strictly in the interface.
abstract method should not contain any of these keywords − private, final, static, native, synchronized [pfsns]	methods in interface should not contain any of these keywords − protected, private, final, static, native, synchronized [ppfsns]
methods are not implicitly public	methods are implicitly public even if not specified (be careful when overriding)
is an incomplete class	specification or prescription for behavior
can extend only one parent class	can implement several interfaces atonce
can have constructors (should contain body)	interfaces can't have constructors

method declarations (including main method):

- The main() method starts the execution of a class for applications and not for applets.
- It should be declared inside a public class.

signature of main() :

valid declarations :

```
public static void main(String argv[]) {
//method body
}

public static void main(String[] argv) {
//method body
}

static public void main(String[] arg) {
//method body
}
```

invalid declaration : This will not compile

```
public void static main(String[] argv) {
//method body
}
```

variable declarations :

Local variables :

- Should be initialized.

Member variables :

- Need not be initialized

```
public class test {
int a; //Member variable
public static void main(String arg[]) {
int b;
int c=0; //correct way of declaring a local
variable
System.out.println(b);//Local variable. Gives
error while compiling "variable b may not have
been initialized"
}
}
```

identifiers :

- A legal identifier is a sequence of letters and digits of unlimited length.
- First character must be a letter. (underscore and dollar sign ($) considered as a letter)
- Subsequent characters may be a letter, dollar sign ($) underscore (_) or a digit.
- The identifier ***can't*** contain a space or #.

eg,
legal identifiers :

test1
$test
_test
_234
$$

illegal identifiers :

2test
my test
my#test

---

10) State the correspondence between index values in the argument array passed to a main method and command line arguments.

```
public static void main(String[] arg) { .....
}
```

The "*main*" function receives an array of String as command line arguments.

```
java test a b c
```

where,
test is the filename (test.java)
arg[0] will contain a

arg[1] will contain b
arg[2] will contain c

Remember : In java, arrays start numbering from 0.

---

11) Identify all Java programming language keywords.

abstract	default	goto(reserved)	null (reserved)	synchronized
boolean	do	if	package	this
break	double	implements	protected	throw
byte	else	import	private	throws
case	extends	instanceof	public	transient (reserved)
catch	final	int	return	try
char	false (reserved)	interface	short	void
class	finally	long	static	volatile
const (reserved)	float	native	super	true(reserved)
continue	for	new	switch	while

---

12) State the effect of using a variable or array element of any kind when no explicit assignment has been made to it.

Automatic (or) local variables :
are not initialized by the system; must be explicitly initialized before being used.

```
public class test {
public static void main(String arg[]) {
int i;
System.out.println("i : " + i);
}
}
```

Compile Error: Variable i may not have been initialized.
System.out.println("int : " + i);

Member (or) Instance variables :
are assigend an initial value automatically.

```
public class initial {
char c;
int in;
short s;
byte b;
long l;
float f;
```

```
double d;
boolean bool;
String str;

public static void main(String arg[]) {
initial i = new initial();
System.out.println("char : " + i.c);
System.out.println("int : " + i.in);
System.out.println("short : " + i.s);
System.out.println("byte : " + i.b);
System.out.println("long : " + i.l);
System.out.println("float : " + i.f);
System.out.println("double : " + i.d);
System.out.println("boolean : " + i.bool);
System.out.println("string : " + i.str);
}
}
```

ouput :

char :
int : 0
short : 0
byte : 0
long : 0
float : 0.0
double : 0.0
boolean : false
string : null

Element of an array :

Array element will be automatically set to default value both inside class and method.

```
public class arr {
int i[] = new int[5];
public static void main(String str[]) {
arr newarr = new arr();
String a[]=new String[10];
System.out.println("String : " + a[0]);
System.out.println("int : " + newarr.i[2]);
}
}
```

output :

String : null
int : 0

---

13) State the range of all primitive data types and declare literal values for String and all primitive types using all permitted formats bases and representations.

Range of primitive data types :

Data type	*bits*	*range (base 2)*	*range (decimal)*
byte	8	-2^7 to $2^7 - 1$	−128 to 127
short	16	-2^{15} to $2^{15} - 1$	−32768 to 32767
int	32	-2^{31} to $2^{31} - 1$	−2147483648 to 2147483647
long	64	-2^{63} to $2^{63} - 1$	−9223372036854775808 to 9223372036854775807
char	16	0 to $2^{16} - 1$	0 to 65536 (or) '\u0000' to '\uffff'

Data type	*bits*
boolean	1 (true/false)
float	32
double	64

literal values for String and all primitive types :

- A literal is a simple value where "what you type is what you get." Numbers, characters, and strings are all examples of literals.
- The integer types are for numbers without fractional parts for eg, 5 is a decimal integer literal of type int. Negative integers are preceded by a minus sign like, −5.
- When a literal value is assigned to a byte or short variable, no error is generated if the literal value is within the range of the target type. Also, an integer literal can always be assigned to a long variable. However, to specify a long literal, you will need to explicitly tell the compiler that the literal value is of type long. You do this by appending an upper− or lowercase L to the literal. eg, 40L.
- Integers can be expressed as octal or hexadecimal: Octal values are denoted in Java by a leading zero. Normal decimal numbers cannot have a leading zero. Thus, the seemingly valid value 09 will produce an error from the compiler, since 9 is outside of octal's 0 to 7 range. eg, 077. A leading 0x (or 0X) denotes an hexadecimal. The range of a hexadecimal digit is 0 to 15, so A through F (or a through f) are substituted for 10 through 15. eg, 0xAF.
- The floating−point literals denotes numbers with fractional parts. There are two types of floating−point types. They are float and double. Floating−point literals in Java default to double precision. To specify a float literal, you must append an F or f to the constant. You can also explicitly specify a double literal by appending a D or d. For eg, 3.55F.
- You can use exponents in floating−point literals using the letter e or E followed by the exponent (which can be a negative number): 11e35 or .35E−2.
- Boolean literals consist of the keywords true and false. The values of true and false do not convert into any numerical representation.
- A literal character is represented inside a pair of single quotes. All of the visible ASCII characters can be directly entered inside the quotes, such as 'a', 'z', and '@'. For characters that are impossible to enter directly, there are several escape sequences, which allow you to enter the character you need. Characters are stored as 16−bit Unicode characters. Unicode characters are expressed in terms of hexadecimal encoding scheme. For eg, \u000d represents a carriage return.
- A String literal is represented by enclosing a sequence of characters between a pairof double quotes. Java doesn't have a built−in string type. Instead, the standard Java library contains a predefined class

called *String*. String literals are expressed by series of characters inside double quotes: "My name is String literal."

Converting from Decimal to Binary, Octal, Hexadecimal
eg,
Decimal to Binary :
18 in Decimal
2|18
−−−−
2|9−0
−−−−
2|4−1
−−−−
2|2−0
−−−−
1−0
10010 in binary

converting negative decimal numbers to binary :
This is by Kathy Kozel in Java Study Group

The left−most bit is the sign bit. If the left most bit is 1 then the number is negative else the number is positive.
if you start with a negative binary number...
1111 1010 // and you want to know the value
1. Flip the bits
0000 0101
2. Add 1
0000 0101
+
0000 0001
−−−−−−−−−−−−
0000 0110 // this is 6, so the value was −6.
By the way, you WILL NOT, repeat WILL NOT be asked to take some big number and represent it negatively in binary. That's what calculators or code are for. But small numbers you may be asked to represent in binary, octal, and hex.

Decimal to Octal :
8|18
−−−−
2−2
22 in octal
Decimal to HexaDecimal :
16|26
−−−−
1−A
1A in Hexadecimal

Converting from Binary, Octal, Hexadecimal to Decimal

10010 in binary
0*2^0 + 1*2^1 + 0*2^2 + 0*2^3 + 1*2^4

0+2+0+0+16 = 18 in decimal
22 in octal
2*8^0 + 2*8^1
2+16=18 in decimal
1A in hexadecimal
A's equivalent in decimal is 10 + 1*16^1 =10 + 16=26 in decimal

---
