

Object Oriented Programming

Series 12

1. A strand of deoxyribonucleic acid (better known as DNA) can be characterized by a finite sequence of nucleotides. There are commonly 4 different nucleotides represented by the letters A, T, G and C. For example, the sequence AATTGCCGT represents a strand of DNA.

Knowing that a DNA strand can have a very high number of nucleotide occurrences, in order to represent it efficiently, Run-Length Encoding (RLE) compression can be used. RLE compression consists of replacing a series of successive occurrences of the same element into a pair $\langle l, e \rangle$ where l is the number of occurrences and e is the element itself. For example, the sequence AAAAAATTT can be compressed by the sequence of the two pairs $\langle 6, A \rangle$ and $\langle 3, T \rangle$.

We want to implement in Java two classes called FullDNA and CompressedDNA both representing a DNA strand (in other words, the sequence of its nucleotides) respectively in its complete form, and in a compressed form using the RLE method.

Both classes must implement the following interface:

```
public interface DNA
{
    Object getNucleotide(int i);
    int getSize();
}
```

where the method `getNucleotide` returns an object encapsulating the i -th nucleotide of a strand (starting from index 0) and where the method `getSize` returns the length of the sequence characterizing the strand (i.e. how many nucleotide compose this strand).

Additionally, the two required classes must respect the following properties:

- The sequence characterizing a DNA strand must be displayed on the terminal in its complete form.
- Two DNA strands must be comparable thanks to the equivalence mechanism in Java, even if they are not represented in the same manner.
- The two classes must be instantiated with a **String** representing a sequence of nucleotides. We assume this String only contains upper case characters.

You are free to implement any additional classes you deem necessary for your solution. You do *not* need to deal with cloning and packaging in your classes. However, implement custom Exceptions to deal with errors.

2. Answer the following questions, and justify your answer.

- (a) What is the difference between **static link** and **dynamic link** ? To what does it applies to ? Illustrate your answer with a concrete example.
- (b) How can you specify that a variable must not be considered when serializing an object ?
- (c) What are the definitions of **class variables**, **instance variables** and **local variables** ? And for each, where is the value of such a variable stored ?
- (d) What is the **constructor chaining** ? In what context is it applied ? In what order ? And by default which constructors are called when a given class is instantiated ?

3. Given these four pieces of code:

```
package a;
class A
{
    public static int m;
    int n;
}
```

```
package a;
public class B extends A
{
    protected static int o;
}
```

```
package a.b;
import a.*;
public class C extends B
{
    public static int p;
}
```

```
package a.b;
import a.*;
public class D
{
}
```

Say for each of the following statements whether they are true or false and justify your answer.

- (a) The class **C** has access to the variable **n**.
- (b) The class **D** has access to the variable **A.m**.
- (c) The class **D** has access to the variable **B.m**.
- (d) The class **A** has access to the variable **B.o**.
- (e) The class **D** has access to the variable **C.o**.
- (f) The class **A** has access to the variable **C.p**.

4. Consider the following Java class:

```
public class Character
{
    private String name;
    private int HP_MAX;
    private int HP_CURRENT;
    private Vector<Item> inventory;
    private Creature pet;

    (...)
}
```

Modify the class **Character** so that its instances are clonable and serializable.

How is your solution impacted by the fact that the classes **Item** and **Creature** are clonable or not ? Assuming the class **Item** is clonable, does your solution actually performs a deep cloning of the inventory ?

Regarding serialization, what assumption do you have to make about the classes **Item** and **Creature** ?

5. The following interface is defined in the source code of a program:

```
public interface Action
{
    void operation();
}
```

You are asked to program a class **ExecutionMachine** containing a single public class method **void execute(Action a, int n)**. This method should acquire the lock of the object referenced by **a**, then run successively **n** times **a.operation()** in a newly created thread (or not at all if **n ≤ 0**), and then release the lock of the object.