EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours de programmation aux sections IN/SC
Sam J.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

INTRODUCTION A LA PROGRAMMATION

Test Semestre I

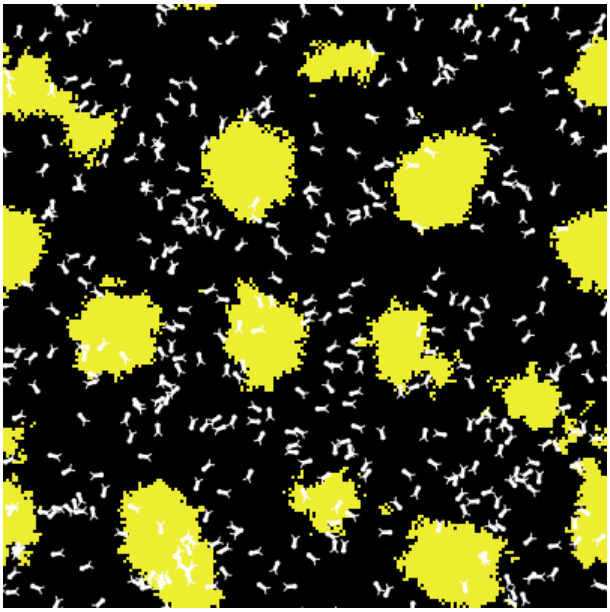
Instructions :

- You have an hour and forty-five minutes to complete this exam (13 :15 - 15 :00) ;
- There are 115 points in total (among which approximately 25 are optional) ;
- This exam is open-book ;
- Please, be sure to **use the appropiate answer sheet for each question**
- This exam consists in 3 independent exercises.
- Exercises do not have the same difficulty. Start with the one that you feel the most comfortable with.

Continue on the other side ☞

Exercise 1 : Conception OO [55 points]

The goal of this exercise is to simulate how termitaries (termite nests) propagate in a environment filled with twigs.



Part of the program is provided in the annex and contains
- a minimalistic model for the simulation environment `Environment` ;
- the `Positionable` class ;
- the `Updatable` interface ;
- the enum type `MovingMode`.

Please, read the entire material for this exercise before writing you answers. Pay particular attention to the comments in the provided material. Note that in this exercise, we do not care about the graphical part of the simulation.

Termitaries are piles of *twigs* (wood sticks) infested with **termites**.
To simplify our simulation, we consider that a termitary is characterized by a unique position : to identify a termite's termitary, you only need to know the termitary's position.

A twig has a position in the environment and is characterized by its length. The position and length are initialized by the constructor.

A termite also has a position in the environment. It is characterized by its *energy level* and must be `Updatable`.

The termite constructor will take as an argument the position of its termitary, which will be used as the termite's initial position. The energy level must be initialized to a given value.

A termite can either randomly move in the environment or move towards a particular target (a `Vector`, as used in your project). We do not care about the implementation details of the termites moves. The only relevant element is that a move towards a target requires to know the position of the final destination.

A termite's general behavior is as follows : it *moves randomly* until it reaches a twig. Upon reaching a twig, the termite picks it up, and *moves in targeted mode* towards its termitary. When this destination is reached, the termite drops the twig and switches back to random movements. At each step, the termite loses a fixed amount of energy. Every time it reaches its termitary, its energy level is increased by a fixed increment. The amount of energy lost while performing a move, and gained upon reaching the termitary

are constants identical for all termites (e.g., 0.5 for a loss of energy, and 1 for a gain). If, while moving randomly, a termite detects that its energy level fell below a fixed threshold (a constant identical for all termites, e.g., 20.0), the termite switches to targeted mode and heads toward its termitary.

Gendered Termites Some of these termites are gendered (male or female). Gendered termites can migrate to another (or a new) termitary. Gendered termites move just like genderless ones, i.e., either randomly or towards a selected target. To simplify, consider that these moves are implemented in exactly the same way, both for random and targeted modes, for gendered and genderless termites. When gendered termites are not in the middle of a migration, they behave as genderless termites. During a migration, they move in targeted mode towards the migration destination.

Reproduction, migration, and death During any iteration of the simulation loop, a termitary can randomly spawn gendered and genderless termites. If a termitary's population increases above a certain threshold (constant and common to all termitaries), all the gendered termites leave the termitary (migration). A common random destination is selected for these termites that start their migration in targeted mode. Upon reaching the destination, a termite replaces its original termitary with this new position. If no termitary exists at this location, a new one is created. Finally, a termite (gendered or not) dies when its energy level reaches 0. Dead termites are simply removed from the simulation.

The goal here is to simulate the evolution of termitaries.

Question 1 : Conception [45 points]

You have to write the class(es) for this simulation that implement the desired functionalities ; only method signatures are required. **You are not asked to write the entire program, or method bodies. We only ask for class attributes and method signatures.** You can describe classes and their content using diagrams or pseudo-code in Java. The following constraints must be satisfied :

1. You can assume that a main program exists, that it creates the `Environment`, and calls its `simulate` method in a loop. This method allows to advance the state of the simulation, i.e., all the entities involved (via the `update` method, as specified by the `Updatable` interface) ;
2. Classes must contain the necessary elements required to implement all the described functionalities, **including the ones that are subtly suggested in the text** (in particular the `update` methods) ;
3. You should rely on **brief comments** to clarify a method functionality or its return types where needed.
4. **Do not forget constructors, or access rights and modifiers.**
5. you are not allowed to use the `protected` modifier for attributes ;
6. your design must avoid code duplication ;
7. only the getters and setters necessary to the simulation must be provided ;
8. You do not have to write import statements ;

Continue on the other side ☞

Question 2 : Programming [10 points]

Based on your design, write the body for the gendered termites `update`. You do not have to provide the code for methods used from other classes.

Exercise 2 : Concepts [37 points]

Provide clear and short answers to the following questions :

1. [**5 points**] Consider the following code :

```
0.  class X {
1.      protected int x;
2.
3.      public X() {
4.          this(100);
5.      }
6.      public X(int x) {
7.          this.x = x;
8.      }
9.  }
10.
11. class Y extends X {
12.     protected int y;
13.
14.     public Y(int x, int y) {
15.         this.x = x;
16.         this.y = y;
17.     }
18. }
```

   (a) Which constructors are executed during the creation of a `Y` instance ?

   (b) Formulate critics about the code. How would you remedy these design issues ?

2. [**4 points**] Provide the output for this code :

```
0.  class Question {
2.      public static void main(String[] args) {
3.          String s1 = new String("Ragging");
4.          String s2 = new String("Bull");
5.
6.          p(s1,s2);
7.
8.          System.out.println(s1);
9.          System.out.println(s2);
10.
11.     }
12.
13.     public static void p(String s1, String s2) {
14.         String tmp;
15.         tmp = s1;
16.         s1 = s2;
17.         s2 = tmp;
18.     }
19. }
```

   Briefly justify you answer.

Continue on the other side ☞

3. [**7 points**] Consider the code below :

```
0.  interface I {}
1.
2.  interface J extends I {}
3.
4.  abstract class A {}
5.
6.  class B extends A implements J {}
7.
8.  class Question {
9.      public static void main(String[] args) {
10.     }
11. }
```

For each of the following instructions (placed separately in the `main` function), indicate if they compile or not :

(a) `A o = new A()`

(b) `A o = new B();`

(c) `B o = (I) new B();`

(d) `A o = new I();`

(e) `I o = new B();`

(f) `J o = new B();`

(g) `J o = (B) new J();`

Provide a brief justification.

4. [**4 points**] Compiling the following code :

```
0.  class Question {
1.      public static void main(String[] args) {
2.          approx(10.);
3.          approx(0.);
4.      }
5.
6.      public static double approx(double x) {
7.          if (x<= 0) throw new Exception("NaN");
8.          return Math.log(x);
9.      }
10. }
```

yields this error message :

```
unreported exception Exception; must be caught or declared to be thrown
```

(a) explain the cause for this error

(b) how would you correct this problem ? (explain what you would add and where)

5. [**3 points**] Can an abstract method be final ? Justify briefly.

6. [**6 points**] Consider this code :

```
0.  interface Drawable {
1.     default void draw() {}
2.     void draw(String color);
3.  }
4.
5.  class Ball implements Drawable {
6.      protected void draw(String color) {}
7.  }
8.
9.  class Question {
10.     public static void main(String[] args) {
11.         Ball ball = new Ball();
12.         Ball.draw();
13.     }
14. }
```

(a) why does it trigger the following compilation error :
`attempting to assign weaker access privileges; was public`

(b) why does it trigger the following compilation error :
`non-static method draw() cannot be referenced from a static context`

(c) Are there still errors if :

i. we delete the declaration of the `draw` method in `ball` (line 6)

ii. if we replace line 11 with `Drawable ball = new Ball();`

Justify briefly.

7. [**4 points**] Why is the following code :

```
List<Double> myList = new ArrayList<>();
```

usually preferred to :

```
ArrayList<Double> myList = new ArrayList<>();
```

Continue on the other side ☞

8. **[4 points]** Consider this program :

```
0.  class Account {
1.      private double amount;
2.      public double setAmount(double m) {
3.          amount = m;
4.          return m;
5.      }
6.  }
7.
8.  class Bank {
9.      List <Account> accounts = new ArrayList<>();
10.
11.     public Bank(List<Account> accounts) {
12.         this.accounts = accounts;
13.     }
14. }
```

(a) Suppose that an account (`Account`) can exist outside of a bank, but that once assigned to a particular one, it becomes its property. What encapsulation issue does this code contain and how would you fix it ? (simply provide an explanation in English).

(b) How would you modify the code to prevent the concept of an account from existing without the concept of a bank ? (give a brief explanation in English).

Exercise 3 : Déroulement de programme [23 points]

The following program compiles and executes without errors.

```
0.  interface I {
1.    default int m1() { return 10;  }
2.    default int m2() { return 5; }
3.  }
4.  abstract class A  {
5.      private int a;
6.      private int b;
7.
8.      public A(int x, int y) { a=x; b=y; }
9.
10.     public A(int x) { this(x,3); }
11.     public A() { this(2); }
12.
13.     public int m1()  { return 2*a; }
14.
15.     public abstract int m2();
16.
17.     public int m3() {
18.         return getB() + m1() + m2();
19.     }
20.     public int getA() { return a; }
21.     public int getB() { return b; }
22. }
23. class B extends A {
24.     private int b;
25.
26.     public B(int x, int y) {
27.         super(x,y) ;
28.         b = 3;
29.     }
30.     public int m1() {
31.       return getB()*getA();
32.     }
33.     public int m2() {
34.         return 5*super.getB();
35.     }
36.     public int getB() { return b; }
37. }
38.
39. class C extends B  {
40.     private int c;
41.     public C(int x , int y , int z) {
42.         super(x,y);
43.         c=z;
44.     }
45.     public int m1() {
46.         return  10*getA();
47.     }
```

```
48.     public int m2() {
49.         return  3*getB()+c;
50.     }
51. } // fin de la classe C
52. class D extends B implements I {
53.     public D() {
54.         super(3,4);
55.     }
56.     public int m1() {
57.         return super.m1() + I.super.m1();
58.     }
59.     public int m2(A a, B b) {
60.         return a.m1() + b.m2();
61.     }
62. }
63. class Deroulement {
64.   public static void main(String[] args) {
65.     int k=0;
66.     B b  = new B(3,5);
67.     A a = b;
68.     System.out.println( ++k + ") " + b.m1());
69.     System.out.println( ++k + ") " + b.m2());
70.     System.out.println( ++k + ") " + a.m1());
71.     System.out.println( ++k + ") " + a.m2());
72.     System.out.println( ++k + ") " + a.m3());
73.     System.out.println( "@@@@@" );
74.     C c = new C(3,4,5);
75.     a = c;
76.     b = c;
77.     System.out.println( ++k + ") " + c.m1());
78.     System.out.println( ++k + ") " + c.m2());
79.     System.out.println( ++k + ") " + a.m1());
80.     System.out.println( ++k + ") " + a.m2());
81.     System.out.println( ++k + ") " + a.m3());
82.     System.out.println( ++k + ") " + b.m1());
83.     System.out.println( ++k + ") " + b.m2());
84.     System.out.println( ++k + ") " + b.m3());
85.     System.out.println( "@@@@@" );
86.     D d = new D();
87.     a = d;
88.     b = d;
89.     System.out.println( ++k + ") " + d.m1());
90.     System.out.println( ++k + ") " + d.m2());
91.     System.out.println( ++k + ") " + d.m2(a,b));
92.     System.out.println( ++k + ") " + d.m2(a,c));
93.     System.out.println( ++k + ") " + d.m2(c,c));
94.     System.out.println( ++k + ") " + a.m3());
95.   }
96. }
```

What is the output ? Briefly justify. **You should explain the steps of the program's execution, without paraphrasing the code.**