COMP-202 – Introduction to Computing 1
Final Exam - Winter 2004 - Version 1

April 22, 2004, from 9:00am to 12:00 noon

Examiners: Ernesto Posse, Yannick Daoudi, Raj Suchak, Associate Examiner: Joseph Vybihal

Last name ______________________________

First Name ______________________________

Id number ______________________________

Section ☐ 1 (Ernesto Posse) ☐ 2 (Yannick Daoudi) ☐ 3 (Raj Suchak) ☐ (Deferred)

Instructions

- No notes, notebooks, textbooks, calculators, cell phones, pagers, laptops or handheld computers are allowed in this exam.
- All answers should be written on the booklet provided. You can write your answers in the back if you need more space, but indicate clearly where are the solutions for each question in sections 2 and 3. **For section 1 (multiple choice) mark your answer in the front page.**
- **Read carefully the questions.**
- Language translation dictionaries are permitted.
- Answers can be given in either English or French were applicable.
- Please return all pages of the exam.

Grading

Section 1: Multiple Choice

Section 1 mark: / 24

Question	Q1	Q2	Q3	Q4	Q5	Q6
Your answer	d	a	c	e	e	a
Mark	/ 2	/ 2	/ 2	/ 2	/ 2	/ 2
Question	Q7	Q8	Q9	Q10	Q11	Q12
Your answer	b	c	a	c	b	d
Mark	/ 2	/ 2	/ 2	/ 2	/ 2	/ 2

Section 2: Problems

Section 2 mark: / 36

Question	Q1	Q2	Q3	Q4	Q5
Mark	/ 5	/ 5	/ 8	/ 9	/ 9

Section 3: Programming

Section 3 mark: / 40

Question	Q1	Q2
Mark	/20	/20

Total: / 100

Section 1: Multiple choice

Question 1 Which of the following is a method call (method invocation)?

a) `class M { int x; }`
b) `boolean m;`
c) `boolean m(int n, char v)`
d) `obj.m(7, 'Z')`
e) `private static double m = 1.618;`

Answer: d)

Question 2 Which of the following is true?

a) A class is made up of methods and attributes (instance variables.)
b) A class does not define a data-type.
c) A method is made up of classes.
d) A class is the value of an object.
e) A method is the value of an object.

Answer: a)

Question 3 Which of the following is true?

a) If B extends A then A will inherit only those methods and attributes of B which are static.
b) If B extends A then B will inherit only those methods and attributes of A which are static.
c) If B extends A and C extends B then C implicitly extends A.
d) If B extends A and C extends A then C implicitly extends B.
e) If B extends A and C extends B then A implicitly extends C.

Answer: c)

Question 4 A private static variable in some class C is accesible in:

a) static methods in C only.
b) non-static methods in C only.
c) private methods in C only.
d) private static methods in C only.
e) in all methods of C.

Answer: e)

Question 5 Which of the following is false?

a) Two different methods may have parameters with the same identifier (name).
b) Two different classes may have methods with the same name.
c) Two different classes may have attributes (instance variables) with the same name.
d) A class may have two (or more) methods with the same name.
e) A class may have two (or more) attributes (instance variables) with the same name.

Answer: e)

Question 6 An object...

a) is a composite piece of data (i.e. a piece of data which may have other pieces of data.)
b) is a primitive data value.
c) is the data type of a class.
d) is a method which does not return anything.
e) is the same as a class.

Answer: a)

Question 7 Consider the following class definition:

```
class SalesPerson {
  private double baseSalary;
  public SalesPerson(double s) { baseSalary = s; }
  public double salary() { return baseSalary; }
  public double salary(int sales) {
    return baseSalary + 0.20 * sales;
  }
  public void raise(double a) {
    baseSalary = baseSalary + a;
  }
}
```

The salary methods above can be characterized as an example of:

a) Method overriding.
b) Method overloading.
c) Polymorphism.
d) Encapsulation.
e) Inheritance.

Answer: b)

Question 8 Assume that you have the class `SalesPerson` from the previous question. Consider the following fragment:

```
SalesPerson annoying, polite;
double v1, v2;
annoying = new SalesPerson(1000.0);
polite = new SalesPerson(1500.0);
v1 = annoying.salary(100);
v2 = polite.salary(200);
polite.raise(500.0);
```

After executing it the values of `v1` and `v2` are

a) 1000.0 and 1500 respectively.
b) 1000.0 and 2000 respectively.
c) 1020.0 and 1540 respectively.
d) 1000.0 and 2040 respectively.
e) 1020.0 and 2040 respectively.

Answer: c)

Question 9 Suppose you have the following class definition:

```
class House {
  int rooms;
  House(int r) { rooms = r; }
  boolean equals(House other) {
    return this.rooms == other.rooms;
  }
}
```

What will be printed after the following is executed?

```
House a, b, c;
a = new House(3);
b = new House(3);
c = a;
if (a == b) System.out.print(1);
if (a == c) System.out.print(2);
if (c == b) System.out.print(3);
if (a.equals(b))  System.out.print(4);
if (a.equals(c)) System.out.print(5);
if (c.equals(b)) System.out.print(6);
```

a) `2456`
b) `123456`
c) `123`
d) `24`
e) `245`

Answer: a)

Question 10 Suppose that class `Paper` is a subclass of (extends) `Document`. Which of the following lines produces a typing (compile-time) error?

```
Document a = new Document();           \\ line 1
Document b = new Paper();              \\ line 2
Paper c = new Document();              \\ line 3
Document e = (Document)(new Paper());  \\ line 4
```

a) 2, 3 and 4
b) 4
c) 3
d) 3 and 4
e) None: all are correct.

Answer: c)

Question 11 Assume that class `E` extends `Exception`. What will the following program fragment print?

```
System.out.print(“a”);
try {
  System.out.print(“b”);
  if (true) throw new E();
  System.out.print(“c”);
}
catch (E v) {
  System.out.print(“d”);
}
System.out.print(“e”);
```

a) `abcde`
b) `abde`
c) `abdce`
d) `abd`
e) `abdc`

Answer: b)

Question 12 What does the following method compute?

```
static int fun(int a, int b)
{
  if (a == 0) return 1;
  else if (a == 1) return b;
  return fun(a - 1, b) * b;
}
```

a) Returns $b!$ (the factorial of `b`, this is, $1 \cdot 2 \cdot 3 \cdot \ldots \cdot (b-1) \cdot b$)
b) Returns $a!$ (the factorial of `a`, this is, $1 \cdot 2 \cdot 3 \cdot \ldots \cdot (a-1) \cdot a$)
c) Returns a^b (`a` to the power of `b`)
d) Returns b^a (`b` to the power of `a`)
e) Returns ab (`a` multiplied by `b`)

Answer: d)

Section 2: Problems

Question 1 For each of the following statements say whether it is true or false:

a) A variable may be used before it is declared.

False

b) A the body of a while loop is executed as long as its condition is true, and no break or return statements are executed in the body.

False (by exceptions)

c) The order in which method definitions appear in a class affects the way the program is executed.

False

d) A protected variable or method in some given class C can be accessed only by the subclasses of C.

False

e) Stacks can be implemented using arrays or linked lists.

True

Question 2 What will the following program print?

```
class A { int u; }
class B {
  static int p(int x, A y, int[] z)
  {
    x = 7;
    y.u = x;
    z[0] = z[1];
    z[1] = 3;
    return z[0];
  }
  public static void main(String[] args)
  {
    int x = 2;
    A y = new A();
    y.u = 3;
    int[] z = { 6, 4 };
    int r = p(x, y, z);
    System.out.println(x);
    System.out.println(y.u);
    System.out.println(z[0]);
    System.out.println(z[1]);
    System.out.println(r);
  }
}
```

Answer:

```
2
7
4
3
4
```

Question 3 Consider the following class definitions:

```
public class Test {
  public static void main(String[] args)
  {
    Rock sedna = new Rock();
    Planet mars =  new Planet();
    sedna.move();    // line 1
    mars.move();     // line 2
    sedna.rotate();  // line 3
    ma rs.rotate();   // line 4
    sedna.rust();    // line 5
    mars.rust();     // line 6
    sedna.shake();   // line 7
    mars.shake();    // line 8
  }
}
class Rock {
  public void move() { System.out.println(“Moving”); }
  public void rust() { System.out.println(“Rusting”); }
  public void shake() { System.out.println(“Shaking”); }
}
class Planet extends Rock {
  public void move() { System.out.println(“Translating”); }
  public void rotate() { System.out.println(“Rotating”); }
  public void shake() { super.shake(); }
}
```

For each of the lines enumerated above (from 1 to 8) say whether it is legal or not, and if it is legal write the output.

Answer:

Line	Legal	Printout if legal
1	yes	`Moving`
2	yes	`Translating`
3	no	
4	yes	`Rotating`
5	yes	`Rusting`
6	yes	`Rusting`
7	yes	`Shaking`
8	yes	`Shaking`

Question 4 Suppose that you have the following class definitions:

```
class Sportsman {
  private boolean active;
  public Sportsman() { active = true; }
  public void accident() { active = false; }
  public void play() { active = true; }
}
class HockeyPlayer extends Sportsman {
  private int teeth;
  public HockeyPlayer() {
    super();
    teeth = 40;
  }
  public int get_teeth() { return teeth; }
  public void play() { teeth--; }
}
class F1Driver extends Sportsman {
  private double salary;
  public F1Driver() { salary = 100000000.0; }
  public boolean is_rich() { return true; }
  public void play() { salary = salary * 1.5; }
}
```

Write a method `training` that takes as input a `Sportsman` and makes it `play` 10 times, and then it has an `accident`. If the sportsman is a `HockeyPlayer`, the method prints the player's number of teeth, and if it is an `F1Driver`, it prints whether the sportsman is rich or not. Keep in mind that `HockeyPlayers` and `F1Drivers` play differently. In a separate method called `test` (with no parameters or return type) create a `HockeyPlayer` and an `F1Driver` and apply the method `training` to both. You can assume that `training` and `test` are in the same class.

```
static void training(Sportsman s)
{
    int game;
    game = 1;
    while (game <= 10) {
        s.play();
        game++;
    }
    s.accident();
    if (s instanceof HockeyPlayer)
        System.out.println(((HockeyPlayer)s).get_teeth());
    if (s instanceof F1Driver)
        System.out.println(((F1Driver)s).is_rich());
}
static void test()
{
    HockeyPlayer p1 = new HockeyPlayer();
    F1Driver p2 = new F1Driver();
    training(p1);
    training(p2);
}
```

Question 5 Suppose that Java did not have the `+`, `-`, `*` and `/` operators, but instead it had two primitive functions `succ` (with signature `int succ(int n)`) and `pred` (with signature `int pred(int n)`) which compute the successor of a number and the predecessor of a number respectively (e.g. `succ(5)` returns `6` and `pred(5)` returns `4`).
Using *only* these functions and the basic constructs of Java (assignment, conditionals, loops, checking for equality, and the `return` statement) write a method `add` (with signature `int add(int a, int b)`) which returns the sum of two numbers (`a` and `b`).

You'll get full points if your solution is recursive, and you'll get 90% if it is not recursive (but correct.) You will get no marks if you use any of the operators `+`, `-`, `*` or `/`, or if you define an entire class or the main method or ask the user for input or print anything. Write *only* the add method.
Hint: For any numbers a and b the following hold: $a + (b + 1) = (a + b) + 1$ and $a + 0 = a$

Answer 1: (recursive)

```
int add(int a, int b)
{
    if (b == 0) return a;
    return succ(add(a, pred(b)));
}
```

Answer 2: (recursive)

```
int add(int a, int b)
{
    if (b == 0) return a;
    return add(succ(a), pred(b));
}
```

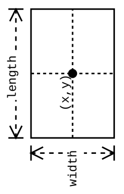
Answer 3: (non-recursive)

```
int add(int a, int b)
{
    int sum;
    sum = a;
    while (b > 0) {
        sum = succ(sum);
        b = pred(b);
    }
    return sum;
}
```

Section 3: Programming

Question 1 In this question you are to write a small program to simulate a simplified radar of a small airport control tower. Let us assume that such a radar can keep track of only two planes.

Each plane can be described by its location, its length and width. You can think of a plane located at (x, y) with length l and width w as a rectangle with these dimensions, and *centered* at (x, y). The radar must be able to detect when the two planes collide. Two planes collide if the rectangles described by the planes intersect.



- Write a class called `Coordinate` that has:
  - Attributes `x` and `y` of type `double`.
  - A constructor that expects two arguments and initializes `x` and `y`.
  - A method `equals` that expects a `Coordinate` as parameter and returns true if and only if the attributes of the parameter are the same as the attributes of the object itself.
- Write a `Plane` class that has:
  - Attributes `length` and `width` of type `double`.
  - An attribute `location` of type `Coordinate` to represent the location (center) of the plane.
  - A constructor that expects three arguments and initializes the attributes.
  - A method `equals` that expects a `Plane` as parameter and returns true if and only if the attributes of the parameter are the same as the attributes of the object itself.
- Write a `RadarScreen` class that has:
  - Two attributes of type `Plane`.
  - A constructor that expects two parameters of type `Plane` and initializes the attributes.
  - A method `collision` which returns true if and only if the two plane objects are different objects and they overlap (i.e. the rectangles intersect.)
- Write a driver class called `ControlTower` which has a main method that creates:
  - `Coordinates` necessary for two planes.
  - Two `Plane` objects for the `RadarScreen` with the given coordinates.
  - A `RadarScreen` with the given plane objects.
  - Asks the `RadarScreen` if there is a collision, and prints "Everything is under control" if there is no collision, or prints "There was a crash!" if there is a collision.

Notes: You may assume any fixed coordinates and dimensions for the planes. You may add additional methods if you need. Make good use of encapsulation.

Hint: To write the collision method it may be easier to think about when two rectangles do not intersect, so that if the rectangles do not intersect, then the collision method returns false and otherwise it returns true.

```
public class ControlTower
{
    public static void main(String[] args)
    {
        Coordinate l1, l2;
        l1 = new Coordinate(5.0, 4.0);
        l2 = new Coordinate(3.0, 6.0);
        Plane p1, p2;
        p1 = new Plane(l1, 3.0, 2.0);
        p2 = new Plane(l2, 2.0, 1.0);
        RadarScreen s = new RadarScreen(p1, p2);
        if (s.collision())
            System.out.println("There was a crash!");
        else
            System.out.println("Everything is under control");
    }
}

class Coordinate
{
    private double x, y;
    public Coordinate(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public boolean equals(Coordinate other)
    {
        return this.x == other.x
            && this.y == other.y;
    }
    public double getX()
    {
        return x;
    }
    public double getY()
    {
        return y;
    }
}
```

```
class Plane
{
    private double length, width;
    private Coordinate location;
    public Plane(Coordinate location, double length, double width)
    {
        this.location = location;
        this.length = length;
        this.width = width;
    }
    public boolean equals(Plane other)
    {
        return this.location.equals(other.location)
            && this.length == other.length
            && this.width == other.width;
    }
    public Coordinate getLocation()
    {
        return location;
    }
    public double getWidth()
    {
        return width;
    }
    public double getHeight()
    {
        return height;
    }
}
```

```
class RadarScreen
{
    private Plane plane1, plane2;
    public RadarScreen(Plane a, Plane b)
    {
        plane1 = a;
        plane2 = b;
    }
    public boolean collision()
    {
        boolean sameplane, no_overlap;
        sameplane = (plane1 == plane2);
        double plane1centerx = plane1.getLocation().getX();
        double plane1centery = plane1.getLocation().getY();
        double plane2centerx = plane2.getLocation().getX();
        double plane2centery = plane2.getLocation().getY();
        double plane1left  = plane1centerx - plane1.getWidth() / 2;
        double plane1right = plane1centerx + plane1.getWidth() / 2;
        double plane2left  = plane2centerx - plane2.getWidth() / 2;
        double plane2right = plane2centerx + plane2.getWidth() / 2;
        double plane1bottom = plane1centery - plane1.getHeight() / 2;
        double plane1top    = plane1centery + plane1.getHeight() / 2;
        double plane2bottom = plane2centery - plane2.getHeight() / 2;
        double plane2top    = plane2centery + plane2.getHeight() / 2;
        no_overlap = plane1right < plane2left
            || plane1left > plane2right
            || plane1top < plane2bottom
            || plane1bottom > plane2top;
        return !sameplane && no_overlap;
    }
}
```

Question 2 Geographical Information Systems are databases which store geographical information. In this question you are asked to develop a small set of classes to represent a set of *geographical items*. We consider only two possible types of items: *mountains* and *lakes*. Any mountain or lake has a name. Mountains in addition, have a height associated to them. Lakes have an area. An *item set* is an unordered collection of *geographical items* with no repetitions (i.e. a mountain does not appear twice in the set.)

Write classes (or interfaces if applicable) to represent geographical items, mountains, lakes and item sets. In particular, geographical item objects must provide a method `print` to print their information, and a method `equals` that decides whether the item is equal to another item (e.g. a mountain is equal to another mountain if it has the same name and height, and a lake is equal to another lake if it has the same name and area.) The class to represent item sets should provide the following methods:

- `isMember`: receives a `name` as parameter and returns true if and only if there is an item in the set with the given `name`.
- `add`: receives an item as parameter and adds it to the set unless there is an equal item in the set already.
- `remove`: receives a `name` as parameter and removes the item with that name from the set.
- `print`: prints the information of each item in the set.
- `equals`: receives another item set `s` as parameter returns true if and only if every item in this set is equal to some item of `s`, and every item of `s` is equal to some item in this set.

You may use arrays or linked lists. You must use the principles of Object-Oriented Programming (encapsulation, inheritance, polymorphism, etc.) There should not be any user input or main method.
Important: The methods above must be written in a way such that they work also if we define new kinds of geographical items (e.g. rivers or forests). For instance the `add` method should be able to add any kind of geographical item, not just mountains and lakes.

```
class GeographicalItem
{
    private String name;
    public GeographicalItem(String name)
    {
        this.name = name;
    }
    public String getName()
    {
        return name;
    }
    public boolean equals(GeographicalItem other)
    {
        return this.name.equals(other.name);
    }
}

class Mountain extends GeographicalItem
{
    private double height;
    public Mountain(String name, double height)
    {
        super(name);
        this.height = height;
    }
    public double getHeight()
    {
        return height;
    }
    public boolean equals(Mountain other)
    {
        return this.height == other.height
            && super.equals(other);
    }
    public void print()
    {
        System.out.println("Mountain: ");
        System.out.println("  Name: " + super.getName());
        System.out.println("  Height: " + height);
    }
}
```

```
class Lake extends GeographicalItem
{
    private double area;
    public Lake(String name, double area)
    {
        super(name);
        this.area = area;
    }
    public double getArea()
    {
        return area;
    }
    public boolean equals(Lake other)
    {
        return this.area == other.area
            && super.equals(other);
    }
    public void print()
    {
        System.out.println("Lake: ");
        System.out.println("  Name: " + super.getName());
        System.out.println("  Area: " + area);
    }
}

class ItemSet
{
    private GeographicalItem[] array;
    private int last;

    public ItemSet()
    {
        array = new GeographicalItem[1000];
        last = -1;
    }

    public boolean isMember(String name)
    {
        int i;
        i = 0;
        while (i <= last) {
            if (name.equals(array[i].getName())) return true;
            i++;
        }
        return false;
    }
    // Continues in the next page
    public void add(GeographicalItem item)
    {
        if (!isMember(item.getName())) {
            if (last == array.length - 1) growArray(100);
            last++;
            array[last] = item;
        }
    }

    public void remove(String name)
    {
        int i;
        i = 0;
        while (i <= last) {
            if (name.equals(array[i].getName())) {
                array[i] = array[last];
                array[last] = null;
                last--;
                break;
            }
            i++;
        }
    }

    private void growArray(int n)
    {
        int new_capacity = array.length + n;
        GeographicalItem[] new_array = new GeographicalItem[new_capacity];
        int i;
        i = 0;
        while (i <= last) {
            new_array[i] = array[i];
            i++;
        }
        array = new_array;
    }

    public void print()
    {
        int i;
        i = 0;
        while (i <= last) {
            array[i].print();
            i++;
        }
    }
```

```
        public boolean contains(GeographicalItem item)
        {
            int i;
            i = 0;
            while (i <= last) {
                if (item.equals(array[i])) return true;
                i++;
            }
            return false;
        }

        public boolean equals(ItemSet s)
        {
            int i;
            i = 0;
            while (i <= last) {
                if (!s.contains(array[i])) return false;
                i++;
            }
            i = 0;
            while (i <= s.last) {
                if (!this.contains(s.array[i])) return false;
                i++;
            }
            return true;
        }
    }
```