1

Nume:
Prenume:
Grupa:

1	2	3		4	5		6	7

Citiți cu atenție toate cerințele înainte de a începe rezolvarea.
Timp total de lucru: 1h30min.

1. **(4p)** Precizați care sunt, în codul de pe verso, clasele, clasele abstracte, interfețele și clasele interne (sau, dacă vreuna din cele patru categorii nu este reprezentată deloc în cod, menționați acest lucru).
2. **(4p)** Ce este suprascrierea? Dar supraîncărcarea? Recunoașteți câte un exemplu pentru fiecare din codul de pe verso.
3. **(4p)** Ce relație se stabilește între BaseObject pe de o parte și ListObject, MapObject și StringObject pe de altă parte? De ce este utilă această relație?
4. **(6p)** Explicați ce este polimorfismul și dați un exemplu elocvent din codul de pe verso.
5. **(6p)** Explicați ce trebuie făcut pentru a introduce un nou tip de obiect în programul de pe verso. Dar pentru a adăuga o nouă operație asupra obiectelor?
6. **(8p)** Care sunt cele cinci principii de design orientat obiect studiate la curs sub acronimul SOLID? Alegeți unul dintre ele și detaliați o situație concretă în care poate fi util (puteți folosi explicații, diagrame și mostre de pseudocod).
7. **(8p)** Explicați o situație în care are sens să folosiți design pattern-ul Singleton, detaliind felul în care ați implementa acest pattern (puteți folosi explicații, diagrame și mostre de pseudocod).

```
abstract class BaseObject {
    public static class ListObject extends BaseObject {
        public final List<BaseObject> objectList;
        public ListObject(List<BaseObject> objectList) {
            this.objectList = objectList;
        }
        @Override
        public void accept(Visitor visitor) {
            visitor.visit(this);
        }
    }
    public static class MapObject extends BaseObject {
        public final Map<String, BaseObject> objectMap;
        public MapObject(Map<String, BaseObject> objectMap) {
            this.objectMap = objectMap;
        }
        @Override
        public void accept(Visitor visitor) {
            visitor.visit(this);
        }
    }
    public static class StringObject extends BaseObject {
        public final String value;
        public StringObject(String value) {
            this.value = value;
        }
        @Override
        public void accept(Visitor visitor) {
            visitor.visit(this);
        }
    }
    public interface Visitor {
        public void visit(ListObject object);
        public void visit(MapObject object);
        public void visit(StringObject object);
    }
    public abstract void accept(Visitor visitor);
}

class PrintVisitor implements BaseObject.Visitor {
    private static String TAB = "    ";
    private static int no_tabs = 0;
    public String getTabs() {
        String res = "";
        for (int i=0;i<no_tabs;i++) {
            res += TAB;
        }
        return res;
    }

    public void visit(BaseObject.ListObject object) {
        int sz = object.objectList.size();
        System.out.println(getTabs()+"ListObject(length=\""+sz+"\")");
        for (BaseObject elem : object.objectList) {
            no_tabs++;
            elem.accept(this);
            no_tabs--;
        }
    }

    public void visit(BaseObject.MapObject object) {
        System.out.println(getTabs()+"MapObject");
        for (Object keyObject : object.objectMap.keySet()) {
            String key = (String) keyObject;
            no_tabs++;
            System.out.println(getTabs()+"Entry(key=\""+key+"\")");
            BaseObject nestedObject = object.objectMap.get(key);
            no_tabs++;
            nestedObject.accept(this);
            no_tabs-=2;
        }
    }

    public void visit(BaseObject.StringObject obj) {
       System.out.println(getTabs()+"String(value=\""+obj.value+"\")");
    }

}

public class Solution {
    private static BaseObject convert(Object element) {
        if (element instanceof JSONArray) {
            JSONArray arrayObject = (JSONArray) element;
            List<BaseObject> list = new ArrayList<>();
            for (Object object : arrayObject) {
                list.add(convert(object));
            }
            return new BaseObject.ListObject(list);
        } else if (element instanceof JSONObject) {
            JSONObject mapObject = (JSONObject) element;
            Map<String, BaseObject> map = new HashMap<>();

            for (Object keyObject : mapObject.keySet()) {
                String key = (String) keyObject;
                Object nestedObject = mapObject.get(key);

                map.put(key, convert(nestedObject));
            }
            return new BaseObject.MapObject(map);
        } else {
            String stringObject = (String) element;
            return new BaseObject.StringObject(stringObject);
        }
    }

    public static void main(String[] args) {
        JSONParser parser = new JSONParser();
        InputStreamReader reader = new InputStreamReader(System.in);
        Object parsed = null;
        try {
            parsed = parser.parse(reader);
        } catch (Exception e) {
            e.printStackTrace();
        }
        BaseObject root = convert(parsed);
        root.accept(new PrintVisitor());
    }
}
```

2

Nume:
Prenume:
Grupa:

1	2	3		4	5		6	7

Citiți cu atenție toate cerințele înainte de a începe rezolvarea.
Timp total de lucru: 1h30min.

1. **(4p)** Precizați care sunt, în codul de pe verso, clasele, clasele abstracte, interfețele și clasele interne (sau, dacă vreuna din cele patru categorii nu este reprezentată deloc în cod, menționați acest lucru).
2. **(4p)** Ce este moștenirea? Recunoașteți un exemplu din codul de pe verso.
3. **(4p)** Argumentați de ce este necesar ca clasele `Add` și `Multiply` să fie abstracte.
4. **(6p)** Ce categorie de clase se folosește pentru a defini obiectele returnate de metodele `getComplexNumberAdder`, `getNumberAdder`, `getNumberMultiplier` și `getComplexNumberMultiplier`? (după ce răspundeți la această întrebare, verificați că ați rezolvat corect primul subiect).
5. **(6p)** De ce este necesară instanțierea unui obiect `AddAndMultiply`? De ce nu se pot apela direct metodele clasei `AddAndMultiply` din metoda main?
6. **(8p)** Care sunt cele cinci principii de design orientat obiect studiate la curs sub acronimul SOLID? Alegeți unul dintre ele și detaliați o situație concretă în care poate fi util (puteți folosi explicații, diagrame și mostre de pseudocod).
7. **(8p)** Explicați o situație în care are sens să folosiți design pattern-ul Factory, detaliind felul în care ați implementa acest pattern (puteți folosi explicații, diagrame și mostre de pseudocod).

```
abstract class Operation{
    public abstract Object perform(Object first, Object second);
    public abstract Object readElement(String stringForm);
    public abstract void printOp(Object first, Object second);
}
abstract class Add extends Operation{
    public void printOp(Object first, Object second){
        System.out.println(first.toString() + "+");
        System.out.println(second.toString() + "=");
        System.out.println(perform(first,second).toString() + "");
    }
}
abstract class Multiply extends Operation{
    public void printOp(Object first, Object second){
        System.out.println(first.toString() + "*");
        System.out.println(second.toString() + "=");
        System.out.println(perform(first,second).toString() + "");
    }
}

class ComplexNumber {
    int re, im;
    public int getRe()           { return re; }
    public int getIm()           { return im; }
    public void setRe(int re)    { this.re = re;     }
    public void setIm(int im)    { this.im = im;     }

    ComplexNumber () {}
    ComplexNumber (int re, int im) {
        this.re = re;
        this.im = im;
    }
    @Override
    public String toString() {
        String result = re + "";
        if (im >= 0) {
            result = result + "+" + im + "i";
        } else {
            result = result + im + "i";
        }
        return result;
    }
}

class AddAndMultiply {

    public Add getComplexNumberAdder() {
       return new Add() {
        public Object perform(Object first, Object second){
            return new ComplexNumber(
             ((ComplexNumber)first).getRe()+
             ((ComplexNumber)second).getRe(),
             ((ComplexNumber)first).getIm()+
             ((ComplexNumber)second).getIm());
        }
        public Object readElement(String stringForm) {
            String[] parts = stringForm.split(",");
            return new ComplexNumber(Integer.parseInt(parts[0]),
                                     Integer.parseInt(parts[1]));
        }
       };
     }

    public Add getNumberAdder() {
      return new Add() {
        public Object perform(Object first, Object second){
            return (Float)first+(Float)second;
        }
        public Object readElement(String stringForm) {
            return Float.parseFloat(stringForm);
        }
      };
    }
    public Multiply getNumberMultiplier(){
      return new Multiply() {
        public Object perform(Object first, Object second){
            return (Float)first*(Float)second;
        }
        public Object readElement(String stringForm) {
            return Float.parseFloat(stringForm);
        }
      };
    }
    public Multiply getComplexNumberMultiplier(){
      return new Multiply() {
        public Object perform(Object first, Object second){
            return new ComplexNumber(
              ((ComplexNumber)first).getRe()*
              ((ComplexNumber)second).getRe()-
              ((ComplexNumber)first).getIm()*
              ((ComplexNumber)second).getIm(),
              ((ComplexNumber)first).getRe()*
              ((ComplexNumber)second).getIm()+
              ((ComplexNumber)first).getIm()*
              ((ComplexNumber)second).getRe());
        }
        public Object readElement(String stringForm) {
            String[] parts = stringForm.split(",");
            return new ComplexNumber(Integer.parseInt(parts[0]),
                                     Integer.parseInt(parts[1]));
        }
      };
    }

    public static void main(String[] args) {
        AddAndMultiply addAndMultiply = new AddAndMultiply();
        HashMap<String, Operation> ops = new HashMap<>();
        ops.put("add complex numbers",
                    addAndMultiply.getComplexNumberAdder());
        ops.put("add numbers",
                    addAndMultiply.getNumberAdder());
        ops.put("multiply complex numbers",
                    addAndMultiply.getComplexNumberMultiplier());
        ops.put("multiply numbers",
                    addAndMultiply.getNumberMultiplier());
        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNext()) {
            Operation operation = ops.get(scanner.nextLine());
            Object first = operation.readElement(scanner.nextLine());
            Object second = operation.readElement(scanner.nextLine());
            operation.printOp(first, second);
        }
    }

}
```