Faculté des Sciences
Aix*Marseille Université

Année universitaire 2015/2016

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☐ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☒ 1er semestre ☐ 2ème semestre ☐ Session 2 Durée de l'épreuve : 2h
Examen de : L3 Nom du diplôme : Licence d'Informatique
Code du module : SIN5U2TL Libellé du module : Programmation et conception orientées objet
Calculatrices autorisées : NON Documents autorisés : OUI, notes de Cours/TD/TP

Exercice 1 : Cases à cocher et boutons radio

Considérons les classes suivantes :

```
public class Checkbox {
  protected boolean selected = false;
  public boolean isSelected() { return selected; }
  public void setSelected(boolean selected) { this.selected = selected; }
  public void doClick() { setSelected(!selected); }
}
```

```
public class Group {
  private List<RadioButton> members;
  public Group() { members = new ArrayList<>(); }
  public void addMember(RadioButton radioButton) { members.add(radioButton); }
  public void clearSelection() { members.forEach(member->member.setSelected(false)); }
}
```

```
public class RadioButton extends Checkbox {
    private Group group;

    public RadioButton(Group group) {
      this.group = group;
      this.group.addMember(this);
    }

    @Override
    public void doClick() {
        if (isSelected()) return;
        group.clearSelection();
        setSelected(true);
    }
}
```

1. Qu'affiche le code suivant ?

```
Group group = new Group();
Checkbox checkbox1 = new RadioButton(group);
Checkbox checkbox2 = new RadioButton(group);
checkbox1.doClick();
checkbox2.doClick();
System.out.println(checkbox1.isSelected()+" "+checkbox2.isSelected());
```

2. Quel principe SOLID est violé par les classes données ci-dessus ? Justifiez votre réponse.

3. Proposez le diagramme de classes d'une nouvelle organisation du code qui corrige ce problème. Vous devez faire en sorte de ne pas dupliquer le code permettant de manipuler le booléen `selected`. On doit également pouvoir écrire le code suivant :

```
Group group = new Group();
Clickable[] elements =
  { new Checkbox(), new RadioButton(group), new RadioButton(group) };
for (Clickable element : elements) element.doClick();
```

4. Implémentez le diagramme de classes donné à la question précédente.
5. La nouvelle organisation du code permet-elle d'écrire le code de la question 1 ? Est-ce satisfaisant ? Justifiez votre réponse.

Exercice 2 : Ligne de commande

Considérons les deux classes suivantes. La classe `CLParser` permet d'analyser les arguments passés en ligne de commande à une application :

```
public class CLParserException extends Exception { }

public class CLParser {
    public enum OptionType { Boolean, String, Integer }
    private List<String> optionNames;
    private List<OptionType> optionTypes;
    private Map<String, Object> values;

    public CLParser() {
        this.optionNames = new ArrayList<>();
        this.optionTypes = new ArrayList<>();
        this.values = new HashMap<>();
    }

    public void addOption(String name, OptionType type) {
        this.optionNames.add(name);
        this.optionTypes.add(type);
    }

    public void parse(String[] args) throws CLParserException {
        for (int index = 0; index < args.length; index++)
          index = parseArg(args, index);
    }

    private int parseArg(String[] args, int index) throws CLParserException {
        String arg = args[index];
        if (!arg.startsWith("-")) throw new CLParserException();
        String name = arg.substring(1);
        if (values.containsKey(name)) throw new CLParserException();
        int position = optionNames.indexOf(name);
        if (position == -1) throw new CLParserException();
        switch (optionTypes.get(position)) {
            case Boolean: values.put(name, true); break;
            case String: index = parseString(name, args, index); break;
            case Integer: index = parseInteger(name, args, index);break;
        }
        return index;
    }
```

```
    public Boolean getBooleanValue(String name) { return values.containsKey(name); }

    public String getStringValue(String name) { return (String) values.get(name); }

    public Integer getIntegerValue(String name) { return (Integer) values.get(name); }

    private int parseString(String name, String[] args, int i) throws CLParserException {
        i++;
        if (i >= args.length)
           throw new CLParserException();
        values.put(name, args[i]);
        return i;
    }

    private int parseInteger(String name, String[] args, int i) throws CLParserException {
        i++;
        if (i >= args.length) throw new CLParserException();
        try {
          int value = Integer.parseInt(args[i]);
          values.put(name, value);
        }
        catch (NumberFormatException e) {
          throw new CLParserException();
        }
        return i;
    }
}
```

Nous avons également la classe suivante :

```
public class MyCommand {
  public static void main(String[] args) {
    CLParser parser = new CLParser();
    parser.addOption("h", CLParser.OptionType.Boolean);
    parser.addOption("n", CLParser.OptionType.Integer);
    parser.addOption("f", CLParser.OptionType.String);
    try {
      parser.parse(args);
      Boolean hValue = parser.getBooleanValue("h");
      Integer nValue = parser.getIntegerValue("n");
      String fValue = parser.getStringValue("f");
      System.out.println(hValue+" "+nValue+" "+fValue);
    } catch (CLParserException e) {
      System.err.println("Erreur");
    }
  }
}
```

1. Quels sont les principes SOLID violés par la classe `CLParser` ? Justifiez votre réponse.
2. Écrivez le code de la classe abstraite `Option`. Les classes qui étendront la classe `Option` représenteront un type d'option possible de la ligne de commande. Elle possède :
   — un attribut final `name` et un constructeur `Option(String name)` qui initialise l'attribut ;
   — la méthode abstraite `int parse(int index, String args[]) throws CLParserException` ;
   — la méthode abstraite et paramétrée `<T> T value(String name)` ;

3. Quelle structure de données peut être utilisée pour retrouver facilement une option à partir de son nom, c'est-à-dire, retrouver une instance d'une classe qui étend `Option` à partir du nom de l'option ?

4. Proposez le diagramme de classes d'une nouvelle organisation du code qui corrige les défauts évoqués à la question 1 et qui utilise la classe abstraite `Option`. Le code suivant doit avoir le même comportement que le code donné en début d'exercice :

```
public static void main(String args[]) {
  CLParser parser = new CLParser();
  parser.addOption(new BooleanOption("h"));
  parser.addOption(new IntegerOption("n"));
  parser.addOption(new StringOption("f"));
  try {
    parser.parse(args);
    Boolean hValue = parser.value("h");
    Integer nValue = parser.value("n");
    String fValue = parser.value("f");
    System.out.println(hValue + " " + nValue + " " + fValue);
  } catch (CLParserException e) { System.err.println("Erreur"); }
}
```

   `Remarque` : Vous avez le droit d'utiliser l'opérateur de "cast" dans les implémentations de la méthode paramétrée `<T> T value(String name)`. Une erreur de "cast" peut se produire si, à l'exécution, on essaie de récupérer la valeur d'un paramètre et de la mettre dans une variable qui ne correspond pas à son type (par exemple, si on fait `Integer fValue = parser.value("f")`). Notez que la value `value` doit retourner `null` si l'option n'est pas présente sur la ligne de commande.

5. Implémentez le diagramme de classes que vous avez donné à la question précédente.

6. Quel principe est violé par un code qui utilise et instancie directement la classe `CLParser` et les classes qui étendent la classe `Option` ? Justifiez votre réponse.

7. Proposez une implémentation de la classe `CLParserBuilder` permettant d'écrire le code suivant à la place du code de la méthode `main` donnée précédemment :

```
public static void main(String args[]) {
  CLParser parser = new CLParserBuilder()
                    .withBooleanOption("h")
                    .withIntegerOption("n")
                    .withStringOption ("f")
                    .build();
  try {
    parser.parse(args);
    Boolean hValue = parser.value("h");
    Integer nValue = parser.value("n");
    String fValue = parser.value("f");
    System.out.println(hValue + " " + nValue + " " + fValue);
  } catch (CLParserException e) { System.err.println("Erreur"); }
}
```