Partie I – Questions de Cours (10 points)

1. Expliquez en quelque phrases à quoi sert Git dans un projet de développement logiciel (1 point)

2. Expliquez en quelque phrases ce que sont les *Fakes*, et dans quelle situation les utiliser (1 point)

3. Expliquez de manière détaillée ce que sont l'inversion et l'injection de dépendances et les intérêts de ces deux techniques (2 point)

4. Dans le code suivant :

```
int score_bowling(int[] rolls) {
   int s = 0;
   int f = 0;
   for (int frame = 0; frame < 10; frame++) {
      if (rolls[f] == 10) {
         s += 10 + rolls[f + 1] + rolls[f + 2];
         f++;
      } else if (rolls[f] + rolls[f + 1] == 10) {
         s += 10 + rolls[f + 2];
         f += 2;
      } else {
         s += rolls[f] + rolls[f+1];
         f += 2;
      }
   }
   return s;
}
```

Expliquez quels éléments contreviennent à des principes de propreté du code, citez ces derniers (2 points).

5. Expliquez ce qu'est une fonction Lambda (1 point)

6. Expliquez le principe de l'opérateur fonctionnel *anyMatch* (1 point)

7. Expliquez de manière détaillée et en donnant des exemples pourquoi il y a très souvent besoin d'utiliser la programmation parallèle lorsqu'on développe un application réseau sur le modèle « Client-Serveur » (2 points).

Partie II –Exercice (10 points)

Le code fourni est un ensemble de classes Java permettant de le prix de cupcakes et de cookies en fonction des toppings que l'ont met dessus. Après avoir examiné le code, proposez une stratégie de refactoring permettant d'améliorer la maintenabilité du code, notamment en réduisant la duplication, et en permettant de rajouter des nouveaux toppings sans modification des classes existantes. Pas besoin d'écrire de code, décrivez simplement de manière détaillée les étapes à suivre ainsi que votre raisonnement et citez les techniques et *design patterns* utilisés.

Classe Cupcake

```
public class Cupcake {
   private final List<TOPPING> toppings;

   public Cupcake(List<TOPPING> toppings) {
      this.toppings = toppings;
   }

   public BigDecimal getPrice() {
      var price = new BigDecimal(2);
```

```
        for (final var topping : toppings) {
            BigDecimal toppingPrice = switch (topping) {
                case CHOCOLATE -> new BigDecimal(0.3);
                case NUTS -> new BigDecimal(0.2);
                case SUGAR -> new BigDecimal(0.1);
            };
            price = price.add(toppingPrice);
        }

        return price;
    }
}
```

Classe Cookie

```
public class Cookie {

    private final List<TOPPING> toppings;

    public Cookie(List<TOPPING> toppings) {
        this.toppings = toppings;
    }

    public BigDecimal getPrice() {
        var price = new BigDecimal(2);

        for (final var topping : toppings) {
            BigDecimal toppingPrice = switch (topping) {
                case CHOCOLATE -> new BigDecimal(0.3);
                case NUTS -> new BigDecimal(0.2);
                case SUGAR -> new BigDecimal(0.1);
            };
            price = price.add(toppingPrice);
        }

        return price;
    }
}
```

Enumération TOPPING

```
public enum TOPPING {
    CHOCOLATE,
    NUTS,
    SUGAR
}
```