IUT de Paris Descartes – Base de la Programmation Objet

Travaux Dirigés – Sujet n°3

Objectifs

- Savoir développer et employer une classe simple
- Savoir limiter les méthodes publiques au strict nécessaire

Un jeu de dé

Nous voulons développer un jeu à base de lancers de dé (à 6 faces). Le jeu est particulièrement simple. Deux joueurs s'affrontent et chacun lance un dé à chaque tour de jeu. Le joueur obtenant la valeur la plus grande gagne un point (aucun point n'est distribué en cas d'égalité). Le premier joueur atteignant 5 points gagne la partie.

À faire

1. Il est décidé que notre application doit reposer sur une classe représentant un dé. En vous limitant à ce qui est nécessaire à la réalisation du jeu décrit ci-dessus, énumérez les prototypes des méthodes (publiques) que doit proposer une telle classe.

   Solution: Les actions élémentaires requises par le jeu sont la capacité de lancer un dé et de comparer la valeur de deux dés. Pour permettre des affichages éventuels, une méthode retournant une représentation textuelle d'un dé est aussi proposée.

   ```
   public class Dé {
     public void lancer() {...}
     public boolean estPlusFortQue(Dé d) {...}
     public String toString() {...}
   }
   ```

   Nous pouvons noter que la méthode `estPlusFortQue` ne nécessite qu'un unique dé en paramètre. Ce dernier sera comparé à celui pour lequel la méthode a été invoquée :

   ```
   Dé d1, d2;
   ...
   if (d1.estPlusFortQue(d2))
     ...
   ```

2. Écrire un programme permettant à deux joueurs de s'affronter.

   Solution:

   ```
   public class Appli {
     private static final int MAX = 5;
   ```

```
  public static void main(String[] args) {
    Dé d1 = new Dé(), d2 = new Dé();
    int p1 = 0, p2 = 0;
    while (p1 < MAX && p2 < MAX) {
      d1.lancer();
      d2.lancer();
      if (d1.estPlusFortQue(d2))
        ++p1;
      else if (d2.estPlusFortQue(d1))
        ++p2;
      System.out.println(d1.toString() + " - " + d2.toString());
    }
    System.out.println("le gagnant est le joueur "
        + (p1 == MAX ? "1" : "2"));
  }
}
```

3. Définissez les données caractéristiques d'un dé et programmez les méthodes de la classe. Vous devez vous assurer que toute donnée est systématiquement initialisée.

   Vous pouvez vous reposer sur la classe `Random` de la bibliothèque standard pour tirer aléatoirement des valeurs :

```
import java.util.Random;

public class Exemple {
  public static void main(String[] args) {
    Random r = new Random();
    final int MAX = 10;
    int i;
    // i est initialisé à une valeur aléatoire dans l'intervalle [0, MAX-1]
    i = r.nextInt(MAX);
  }
}
```

Solution: Plusieurs solutions sont possibles concernant l'initialisation de la valeur du dé. Nous avons choisi de lui affecter une valeur aléatoire. Une valeur constante comprise entre 1 et 6 est aussi possible. Une valeur particulière (telle que 0) indiquant que le dé n'a pas encore été lancé est aussi possible. Toutefois, pour cette dernière solution, les méthodes `estPlusFortQue` et `toString` devraient lever une exception si un dé employé n'a pas été lancé au préalable. Un objet doit toujours avoir un état cohérent et les méthodes doivent former un tout logique.

```
public class Dé {
  private static final int NB_FACES = 6;
  private static final Random rd = new Random();
  private int valeur;

  public Dé() {
    lancer(); // équivalent à this.lancer()
  }

  public void lancer() {
```

```
    valeur = 1 + rd.nextInt(NB_FACES);
  }

  public boolean estPlusFortQue(Dé d) {
    return valeur > d.valeur;
  }

  public String toString() {
    return Integer.toString(valeur);
  }
}
```

4. Pour coller à la réalité, il est décidé que le lancement d'un dé peut conduire à ce que celui-ci soit *cassé* (c'est à dire qu'il s'arrête sur un obstacle faisant qu'il ne soit pas bien à plat). Un dé dans une telle situation ne doit pas pouvoir être comparé à un autre dé. Par contre, il doit être possible de l'afficher (via `toString`). Sachant qu'il y a une chance sur cent qu'un dé soit cassé à l'issu d'un lancement, intégrez cette modification à la classe.

Solution: La non satisfaction de la précondition de la méthode `estPlusFortQue` est signalée ici par une levée d'exception. L'emploie de `assert` est tout aussi acceptable. Le fait que cette méthode dispose d'une précondition nous impose de proposer une méthode permettant à l'utilisateur de la classe de vérifier qu'il ne va pas la violer. C'est le rôle de la méthode `estCassé`.

```
public class Dé {
  private static final int NB_FACES = 6;
  private static final int CASSE = 0, PROBA_CASSE = 100;
  private static final Random rd = new Random();
  private int valeur;

  public Dé() {
    lancer();
  }

  public void lancer() {
    if (rd.nextInt(PROBA_CASSE) == 0)
      valeur = CASSE;
    else
      valeur = 1 + rd.nextInt(NB_FACES);
  }

  public boolean estCassé() {
    return valeur == CASSE;
  }

  public boolean estPlusFortQue(Dé d) {
    if (estCassé() || d.estCassé())
      throw new RuntimeException("au moins un des deux dés est cassé");
    return valeur > d.valeur;
  }

  public String toString() {
    if (estCassé())
      return "cassé";
```

```
    return Integer.toString(valeur);
  }
}
```

5. Corrigez en conséquence le programme principal.

Solution:

```
public class Appli {
  private static final int MAX = 5;

  public static void main(String[] args) {
    Dé d1 = new Dé(), d2 = new Dé();
    int p1 = 0, p2 = 0;
    while (p1 < MAX && p2 < MAX) {
      do {
        d1.lancer();
      } while (d1.estCassé());
      do {
        d2.lancer();
      } while (d2.estCassé());
      if (d1.estPlusFortQue(d2))
        ++p1;
      else if (d2.estPlusFortQue(d1))
        ++p2;
      System.out.println(d1.toString() + " - " + d2.toString());
    }
    System.out.println("le gagnant est le joueur "
        + (p1 == MAX ? "1" : "2"));
  }
}
```