ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Kata Bowling avec IntelliJ

Développement dirigé par les tests avec Intellij IDEA

1. Kata Bowling[1]

Une partie de bowling compte dix *carreaux* (*frames en anglais*). Chaque joueur lance deux boules à chaque carreau, sauf en cas de *strike*. Un *strike* consiste à faire tomber les dix quilles (*pins en anglais*) avec la première boule. Le *spare* consiste à faire tomber les dix quilles avec les deux tirs consécutifs du carreau.

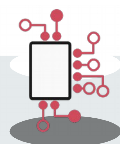
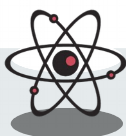
Le score de chaque joueur est le nombre total de quilles qu'il a abattues plus les bonus donnés par les *strikes* ou les *spares*.

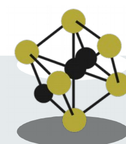
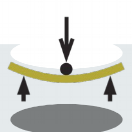
- Pour un jeu ouvert, le score d'un carreau est simplement le nombre de quilles abattues pour les deux lancers.
- En cas de *strike*, le score est donc de 10 plus un bonus égal au nombre de quilles abattues après les deux lancers suivants.
- En cas de *spare*, le score est de 10 plus un bonus égal au nombre de quilles abattues au lancer suivant.

Ainsi, la marque parfaite est de 300 points, pour douze *strikes* consécutifs.

Le dixième jeu est particulier. En cas de *strike* au premier lancer, deux lancers supplémentaires sont accordés. En cas de réalisation d'un *spare*, un lancer supplémentaire est accordé. Au maximum, 21 lancés sont possibles.

1 http://butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata

1.1 Créer la classe de test unitaire Game

Fichier test	Fichier source
	Créer le paquet `fr.ensicaen.ecole.gl.bowling` en une seule fois. Pour cela, cliquer sur l'icône `src` de l'explorateur, puis **Alt+Insert** puis taper le nom du paquet entier. `public final class Game { }`

1.2 Créer la classe de test associée

`public final class GameTest { }`	Créer la classe de test par **Ctrl+Shift+T**.

1.3 Créer la méthode de test : *testGutterGame()* (score = 0)

Ajouter la première méthode test par **Alt+insert** :
`@Test`
`public void testGutterGame {`
Ajouter le code :
`Game g = new Game();`
Ctrl+space après chaque caractère de début de mot pour la complétion.
Taper **fori+Tab** pour produire la boucle :
`for (int i = 0; i < 20; i++) {`
`g.roll(0);`
`}`
Ajouter automatiquement la méthode `roll()` dans la classe Game par **Alt+Entrée**.
Ctrl+B sur la méthode `roll()` pour aller à sa définition dans la classe source.	

	`public void roll( int i ) {`
	`}`
	Ctrl+E pour revenir vers la classe de test.

Taper `a`, puis utiliser la complétion de code pour ajouter : `assertEquals(0, g.score());` Ajouter automatiquement la méthode `score()` par **Alt+Entrée**.	

	`public int score() {`
	`return -1;`
	`}`
	Exécution de la méthode de test :
	Ctrl+Shift+F10 → ROUGE

	`public int score() {`
	`return 0;`
	`}`
	Exécution de tous les tests : **Shift+F10 : VERT**

1.4 Refactoring

Renommer `i` en `pins` :

```
public void roll( int pins )
```

1.5 Créer la méthode *testAllOnes()* (score=20)

Alt+insert test method :

```
@Test
public void testAllOnes() {
  Game g = new Game();
```

fori+tab

```
  for (int i = 0; i < 20; i++) {
    g.roll(1);
  }
```

a+complétion

```
  assertEquals(20, g.score());
}
```

Exécution : **Ctrl+Shift+F10 → ROUGE**
Ctrl+E pour aller dans le source.

```
private int _score = 0;
public void roll( int pins ) {
  _score += pins;
}
public int score() {
  return _score;
}
```

Commencer par créer le contenu de la méthode `roll()`, puis régler la création de l'attribut `_score` avec **Alt+Enter**.
Exécution Shit+F10 → VERT

1.6 Refactoring

```
private Game _game;
@Before
protected void setUp() { // Fixture
  _game = new Game ();
}
public void testGutterGame() {
  rollMany(20, 1);
  assertEquals(20, _game.score());
}
private void rollMany(int n, int pins) {
  for (int i = 0; i < n; i++) {
    _game.roll(pins);
  }
}
```

Ctrl+Alt+F pour transformer `Game` en attribut.
Alt+Insert pour créer la méthode `setUp()`.
Ctrl+Alt+V pour créer les variables `pins` et `n`. À ne faire que sur la première méthode, la seconde sera faite automatiquement.
Ctrl+Alt+M pour construire la méthode `rollMany()`.

Ctrl+Alt+N Remplacer les variables `pins` et `n` par leur valeur.
Shit+F10 → VERT

1.7 Créer la méthode *testOneSpare()* (score=16)

```
@Test
public void testOneSpare() {
  _game.roll(5);
  _game.roll(5); // spare
  _game.roll(3);
  rollMany(17,0);
  assertEquals(16,_game.score());
}
```

Ctrl+D permet de dupliquer une ligne.
Shit+F10 → ROUGE
Ctrl+B sur la méthode `score()` permet de naviguer dans le code de la méthode.

La conception est fausse.
Les responsabilités sont mal placées.
Ctrl+E pour revenir au code du test.

Commenter la méthode de test pour faire du refactoring : **Ctrl+W** plusieurs fois dans la méthode à commenter pour sélectionner toutes les lignes puis **Ctrl+/** pour commenter.
Shift+F10 → VERT

```
public class Game {
  private int _rolls[] = new int[21];
  private int _currentRoll;

  public void roll( int pins ) {
    _rolls[_currentRoll++] = pins;
  }

  public int score() {
    int score = 0;
    for (int i = 0; i < _rolls.length;
i++) {
      score += _rolls[i];
    }
    return score;
  }
}
```

Commencer par la ligne :

```
    _rolls[_currentRoll++] = pins;
```

Alt+Enter sur `_currentRoll` pour créer l'attribut correspondant.
F2 pour naviguer vers la deuxième erreur.
Alt+Enter sur `_rolls` pour résoudre le problème.
Shift+F10 : VERT

Ctrl+/ Dé-commenter
Shift+F10 → ROUGE

Cela ne va toujours pas fonctionner parce qu'on ne représente pas la notion de *Frame*.

Ctrl+/ Re-commenter le dernier test
Ctrl+E pour revenir dans le source

```
public int score() {
  int score = 0;
  int i = 0;
  for (int f = 0; f < 10; f++)  {
```

ENSICAEN - Spécialité Informatique

<table>
<tr><td></td><td><pre>   score += _rolls[i] + _rolls[i+1];
    i += 2;
  }
  return score;
}</pre>Shift+F10 : VERT</td></tr>
<tr><td>Re-commenter → ROUGE</td><td><pre>public int score() {
  int score = 0;
  int i = 0;
  for (int f = 0; f < 10; f ++) {
    if (_rolls[i] + _rolls[i+1]==10) {
      // spare
      score += 10 + _rolls[i + 2];
      i += 2;
    } else {
      score +=_rolls[i] + _rolls[i+1];
      i += 2;
    }
  }
  return score;
}</pre>Ctrl+Alt+T sur la ligne <code>_rolls[i]+_rolls[i+1]</code> permet de l'entourer d'un <code>if</code>.</td></tr>
<tr><td>Refactoring</td><td><pre>public int score() {
  int score = 0;
  int ball = 0;
  for (int f = 0; f < 10; f++) {
    if (isSpare(ball)) {
      score += spareBonus(ball);
      ball += 2;
    } else {
      score += _rolls[ball] +
               _rolls[ball + 1];
      bal += 2;
    }
  }
  return score;
}
private boolean isSpare(int ball) {
  return _rolls[ball]
       + _rolls[ball + 1] == 10;
}
private int spareBonus(int ball) {
  return 10 + _rolls[ball + 2];
}</pre>Shift+F10 → VERT</td></tr>
<tr><td><pre>private void rollSpare() {
  _game.roll(5);
  _game.roll(5);
}</pre>Ctrl+Alt+M pour construire la méthode <code>rollSpare()</code> à partir du code.
Ctrl+B sur l'appel de <code>rollSpare()</code> pour retourner au code source et supprimer le commentaire <code>//spare</code>.
VERT</td><td></td></tr>
</table>

1.8 Créer la méthode *testOneStrike()* (score=24)

```
@Test
public void testOneStrike() {
  _game.roll(10); // strike
  _game.roll(3);
  _game.roll(4);
  rollMany(17, 0);
  assertEquals(24, _game.score());
}
```

```
if (isStrike(ball)) {
  score += 10 + strikeBonus(ball);
  ball++;
}
```

VERT

Refactoring

```
public int score() {
  int score = 0;
  int ball = 0;
  for (int f = 0; f < 10; f++) {
    if (isStrike(ball)) {
      score += 10 + strikeBonus(ball);
      ball++;
    } else if (isSpare(pin)) {
      score += 10 + spareBonus(ball);
      ball += 2;
    } else {
      score += sumOfpinsInFrame(ball);
      ball += 2;
    }
  }
  return score;
}
private int sumOfpinsInFrame(int ball){
  return _rolls[ball]+_rolls[ball+1];
}
private int spareBonus(int ball) {
  return _rolls[ball + 2];
}
private int strikeBonus(int ball) {
 return _rolls[ball+1]+_rolls[ball+2];
}
private boolean isStrike(int ball) {
  return _rolls[ball] == 10;
}
```

```
private void rollStrike() {
  _game.roll(10);
}
```

1.9 Créer la méthode *testPerfectGame()* (score=300)

```
@Test
public void testPerfectGame() {
  rollMany(12,10);
  assertEquals(300, _game.score());
}
```

Shift+F10 → VERT

Mettre 301, ressayer pour vérifier que l'on n'a pas fait de « happy test », **Shift+F10 → ROUGE** puis remettre 300, **Shift+F10 → VERT**	