Ficha 3 - refatoring PA 2020/21

```
/* Snippet 1*/
public class Employee {
    int INTERN = 1;
    int FULLTIME = 2;
    int PARTTIME = 3;
    //...
    double getSalary() {
        switch(type) {
            case INTERN:
            // Long calculation 1
            return res1;
            case FULLTIME:
            // Long calculation 2
            return res2;
            case PARTTIME:
            // Long calculation 3
            return res3;
        }
    throw new RuntimeException();
    }
}
```

Relativo ao código em *Snippet 1*

P1 - Qual das técnicas de refactoring é a mais adequada para lidar com os vários casos no método getSalary()?

1. Extract method
2. Replace conditional with polymorphism
3. Replace delegation with heritance.
4. Nenhuma das anteriores

```
/* Snippet 2*/
    public int getScore1() {
        int res;
        res = (int)(Math.random()*6) + 1;
        dice[0].setFaceValue(res);
        res = (int)(Math.random()*6) + 1;
        dice[1].setFaceValue(res);
        int score = dice[0].getFaceValue() +
        dice[1].getFaceValue();
        return score;
    }
    public int getScore2() {
        int res;
        res = rollDie();
        dice[0].setFaceValue(res);
```

```
        res = rollDie();
        dice[1].setFaceValue(res);
        int score = dice[0].getFaceValue() +
        dice[1].getFaceValue();
        return score;
    }
```

Relativo ao código em *Snippet 2*

P2 - Qual das técnicas de refactoring foi utilizada no método getScore1 de modo a obter o método getScore2?

1. Extract method
2. Inline method
3. Remove duplicated code
4. Nenhuma das anteriores

```
/* Snippet 3*/
    public class Student {
        String name;
        int id;
        // Student methods
    }
    public class Course {
        ArrayList<Student> studentList;
        // Course methods
    }

    public void main() {
        Student s1 = new Student("Ana");
        Student s2 = new Student("Zé");
        Course pa = new Course();
        pa.studentList.add(s1);
        pa.studentList.add(s2);
}
```

Relativo ao código em *Snippet 3*
P3 - Qual o BAD SMELL detetado no código apresentado ?

1. Refused Bequest
2. Middle man
3. Inappropriate intimacy
4. Nenhuma das anteriores

P4 - Assinale a afirmação incorreta?

1. Antes de iniciar o refactoring a um módulo de código, deve existir uma suite de testes.
2. O refactoring influencia (e modifica) o comportamento externo de uma classe.
3. O refactoring influencia (e modifica) a legibilidade do código.
4. O refactoring influencia (e modifica) a arquitectura interna de uma classe.

P5 - Qual das seguintes actividades deve ser considerada como refactoring?

1. Adição de novas funcionalidades.
2. Melhorar a performance da aplicação.
3. Implementação de uma suite de testes.
4. Nenhuma das anteriores.

```
/* Snippet 4*/
    public void imprimeCondutoresInicial(char inicial) {
        for (Condutor condutor : listCondutores) {
            if (condutor.getNome().charAt(0) == inicial) {
                System.out.println(" Condutor " + condutor.getNome());
                System.out.println(" Camiao " +
condutor.getCamiao().getMatricula().getId());
            }
        }
    }
```

Relativo ao código em *Snippet 4*
P6 - Qual o BAD SMELL detetado no código apresentado ?

1. Refused Bequest
2. Message chain
3. Long Method
4. Nenhuma das anteriores

Relativo ao código em *Snippet 4*
P7 - Qual a técnica mais adequada para lidar com o Bad SMELL apresentado ?

1. Extract Class
2. Replace Bidirecional Association with Unidirecional Association
3. Hide Delegation
4. Nenhuma das anteriores

```
/* Snippet 5*/
    public class Circle {
    private Point center;
    private int radius;

    public Circle() {
        this.center = new Point(0,0);
        this.radius = 1;
    }
   //getters e setters
    @Override
    public String toString() {
        return "Circle{" +
                "center=" + center +
                ", radius=" + radius +
                '}';
    }
```

```
}

public class Main {

    public static void main(String[] args) {
      Circle c= new Circle();
      for(int i=0; i< 5;i++) {
          c.setCenter(new Point(c.getCenter().getX()+1, c.getCenter().getY()+1));
          System.out.println(c);
      }
   }
}
```

```
/* Snippet 5.1*/
public static final int REPEAT=5;
    public static void main1(String[] args) {
        Circle c= new Circle();
        for(int i=0; i< REPEAT;i++) {
            c.setCenter(new Point(c.getCenter().getX()+1, c.getCenter().getY()+1));
            System.out.println(c);
        }
    }
}
```

```
/* Snippet 5.2*/
public class Circle {
    private Point center;
    private int radius;

    public Circle() {
        this.center = new Point(0,0);
        this.radius = 1;
    }

    public void moveCenter(int i, int i1) {
        Point p= getCenter();
        p.setX(p.getX()+i);
        p.setY(p.getY()+i);
        setCenter(p);
    }

    @Override
    public String toString() {
        return "Circle{" +
                "center=" + center +
                ", radius=" + radius +
                '}';
    }
}
public class Main {
   public static void main(String[] args) {
        Circle c= new Circle();
        for(int i=0; i< 5;i++) {
```

```
            c.moveCenter(1,1);
            System.out.println(c);
        }
    }
```

```
 /* Snippet 5.3*/
public class Main {
    public static void main3(String[] args) {
        Circle c= new Circle();
        for(int i=0; i< 5;i++) {
            c.setCenter(new Point(c.getCenter().getX()+i, c.getCenter().getY()+i));
            System.out.println(c);
        }
        }
    }
}
```

Relativo ao código em *Snippet 5* e às variantes 5.1, 5.2 e 5.3
P8 - Qual dos fragmentos de código **não** é o resultado da aplicação de refatoring?

1. Snippet 1
2. Snippet 2
3. Snippet 3
4. Todos conrespondem à aplicação de uma ou mais tecnicas de refatoring