'Code smells'

Refaktoryzacja polega na poprawianiu organizacji i polepszaniu czytelności kodu źródłowego.

Ogólnie chodzi o poprawienie cech jakości kodu źródłowego i likwidowanie „code smells”- niezręcznych konstrukcji programistycznych.

Za i przeciw refaktoryzacji.

- **Stosujemy kiedy:**

- 1. Zmieniliśmy coś w kodzie (three strikes and refactor).
- 2. Była zmiana w funkcjonalności.
- 3. Podczas naprawiania błędu (fixing a bug).
- 4. Podczas inspekcji kodu.

- **Nie stosujemy, gdy:**

- 1. Mamy mało czasu (at close deadlines).

1.2. Niestabilny kod.

Rodzaje 'code smells':

1. Powtórzenie kodu.
2. Zbyt wiele komentarzy.
3. 'Type code'.
4. Obszerne klasy.
5. Długie listy parametrów.
6. Metody zawierające zbyt wiele kodu.
7. Niektóre zmienne nie zawsze są stosowane.

Przykład 1:

```
class Shape {
    final static int TYPELINE = 0;
    final static int TYPERECTANGLE = 1;
    final static int TYPECIRCLE = 2;
    int shapeType;
    //starting point of the line.
    //lower left corner of the rectangle.
    //center of the circle.
    Point p1;
    //ending point of the line.
    //upper right corner of the rectangle.
    //not used for the circle.
    Point p2;
    int radius;
}
```

```
class CADApp {
void drawShapes(Graphics graphics, Shape shapes[]) {
  for (int i = 0; i < shapes.length; i++) {
    switch (shapes[i].getType()) {
        case Shape.TYPELINE:
          graphics.drawLine(shapes[i].getP1(), shapes[i].getP2());
          break;
        case Shape.TYPERECTANGLE:
          //draw the four edges.
          graphics.drawLine(...);
          graphics.drawLine(...);
          graphics.drawLine(...);
          graphics.drawLine(...);
          break;
        case Shape.TYPECIRCLE:
          graphics.drawCircle(shapes[i].getP1(), shapes[i].getRadius());
          break;
        }
      }
    }
  }
```

Problem 1 : Kod będzie w przyszłości zmieniany

```
class Shape {
  final static int TYPELINE = 0;
  final static int TYPERECTANGLE = 1;
  final static int TYPECIRCLE = 2;
  final static int TYPETRIANGLE = 3;
  int shapeType;
  //starting point of the line.
  //lower left corner of the rectangle.
  //center of the circle.
  Point p1;
  //ending point of the line.
  //upper right corner of the rectangle.
  //not used for the circle.
  Point p2;
  //third point of the triangle.
  Point p3;
  int radius;
}
```

```
class CADApp {
void drawShapes(Graphics graphics, Shape shapes[]) {
  for (int i = 0; i < shapes.length; i++) {
    switch (shapes[i].getType()) {
        case Shape.TYPELINE:
          graphics.drawLine(shapes[i].getP1(), shapes[i].getP2());
          break;
        case Shape.TYPERECTANGLE:
          //draw the four edges.
          graphics.drawLine(...);
          graphics.drawLine(...);
          graphics.drawLine(...);
          graphics.drawLine(...);
          break;
        case Shape.TYPECIRCLE:
          graphics.drawCircle(shapes[i].getP1(), shapes[i].getRadius());
          break;
        case Shape.TYPETRIANGLE:
          graphics.drawLine(shapes[i].getP1(), shapes[i].getP2());
          graphics.drawLine(shapes[i].getP2(), shapes[i].getP3());
          graphics.drawLine(shapes[i].getP3(), shapes[i].getP1());
          break;
        }
      }
    }
  }
```

- Dodanie możliwości
- rysowania nowej figury(trójkąt)
- wymusiło modyfikacje obu klas.

'Smell code' nr 1:

- Kod takiego typu, powinien być poważnym ostrzeżeniem ,że w przyszłości mogą pojawić się problemy.

```
class Shape {
  final static int TYPELINE = 0;
  final static int TYPERECTANGLE =1;
  final static int TYPECIRCLE = 2;
  int shapeType;
}
```

Przekształcenie 'type code' na klasy.

- **Problem:**
- Klasa posiada pole o skończonej liczbie wartości, którego wartość nie
- wpływa na zachowanie
- **Cel**
- Przekształcenie pola w nową klasę

1)Mechanika

2)Utwórz nową klasę

3)Dodaj do klasy źródłowej pole typu tej klasy i zainicjuj je

4) dla każdej metody w klasie źródłowej, która korzysta z oryginalnego

1)pola stanu, utwórz jej odpowiednik korzystający z nowego pola

5)zmień metody ,tak aby korzystały z nowych metod

6)skompiluj i przetestuj

```
class Shape {
}
class Line extends Shape {
  ...
}
class Rectangle extends Shape {
  ...
}
class Circle extends Shape {
  ...
}
```

'Smell code' nr 2:

```
class Shape {
  ...
  Point p1;
  Point p2;
  int radius;
}
```

Pojawia się zmienna (int radius),która nie zawsze jest używana.

'Smell code' nr 3:

```
class Shape {
  ...
  Point p1;
  Point p2;
  int radius;
 }
```

Zmienne p1,p2 powinny mieć inne nazwy ze względu na przejrzystość kodu.

Usunięcie 'smell code' nr1-3

```
class Shape {
}
class Line extends Shape {
    Point startPoint;
    Point endPoint;
}
class Rectangle extends Sha
    Point lowerLeftCorner;
    Point upperRightCorner;
}
class Circle extends Shape {
    Point center;
    int radius;
}
```

Usunięcie 'smell code' nr 1 za pomocą stworzenie podklas,klasy Shape.

Zmienna int radius została dodana do Podklasy Circle dzięki temu zostanie Użyta tylko w potrzebie nie zajmując zbędnie pamięci.

Nazwy zmiennych p1,p2 zostały zmienione na bardziej przejrzyste.

Poprawiony kod.

```
interface Shape {
    void draw(Graphics graphics);
}
class Line implements Shape {
    Point startPoint;
    Point endPoint;
    void draw(Graphics graphics) {
        graphics.drawLine(getStartPoint(),
        getEndPoint());
    }
}
class Circle implements Shape {
    Point center;
    int radius;
    void draw(Graphics graphics) {
```

```
class Rectangle implements Shape {
    Point lowerLeftCorner;
    Point upperRightCorner;
    void draw(Graphics graphics) {
        graphics.drawLine(...);
        graphics.drawLine(...);
        graphics.drawLine(...);
        graphics.drawLine(...);
    }
}
class CADApp {
void drawShapes(Graphics graphics, Shape shapes[]) {
    for (int i = 0; i < shapes.length; i++) {
        shapes[i].draw(graphics);
    }
}
}
```

Klasa **CADApp** została uproszczona,do minimum, dzięki wprowadzeniu interfejsu.

Kod jest przejrzysty,dzięki temu ,że jest więcej mniejszych klas,które są zrozumiałe,a w wypadku jakiś problemów poprawa ich nie powinna stanowić problemu.

Zadanie:

```
class SurveyData {
String path; //save the data to this file.
boolean hidden; //should the file be hidden?
//set the path to save the data according to the type of data (t).
  void setSavePath(int t) {
    if (t==0) { //raw data.
      path = "c:/application/data/raw.dat";
      hidden = true;
    } else if (t==1) { //cleaned up data.
      Path = "c:/application/data/cleanedUp.dat";
      hidden = true;
    } else if (t==2) { //processed data.
      Path = "c:/application/data/processed.dat";
      hidden = true;
    } else if (t==3) { //data ready for publication.
      Path = "c:/application/data/publication.dat";
      hidden = false;
    }
  }
}
```

Przykładowe rozwiązanie:

```
class SurveyDataType {
String baseFileName;
boolean hideDataFile;
SurveyDataType(String baseFileName, boolean hideDataFile){
this.baseFileName = baseFileName;
this.hideDataFile = hideDataFile;
}
String getSavePath(){
return "c:/application/data/" + baseFileName + „.dat";
}
  static SurveyDataType rawDataType = new SurveyDataType("raw",true);
  static SurveyDataType cleanedUpDataType = new SurveyDataType("cleanedUp",true);
  static SurveyDataType processedDataType = new SurveyDataType("processed",true);
  static SurveyDataType publicationDataType = new
  SurveyDataType("publication",false);
}
```

- Dziękujemy za uwagę!