16TIN2054 – Teknik Pemrograman Praktek

Refactoring - Week 14

Dikerjakan oleh:

Muhammad Azhar Alauddin – 201524013

1AD4 Jurusan Teknik Komputer dan Informatika

Tugas ini dikumpulkan untuk memenuhi sebagian persyaratan kelulusan mata kuliah Teknik Pemrograman Praktek

Program Studi D4 Teknik Informatika

Jurusan Teknik Komputer dan Informatika

Politeknik Negeri Bandung

2020/2021

1. Rename a Field

a. Problem

Penamaan field seharusnya disesuaikan dengan peran dan fungsinya. Namun, pada Cell field terdapat nama player yang kurang sesuai dengan peran dan fungsi peran player di dalam class Cell

b. Solution

Mengganti nama field player yang sesuai dengan perannya, yakni owner

c. Before

Cell.java

```java
package edu.ncsu.monopoly;

public abstract class Cell {
  private boolean available = true;
  private String name;
  protected Player player;

  public String getName() {
    return name;
  }

  public Player getPlayer() {
    return player;
  }

  public int getPrice() {
    return 0;
  }

  public boolean isAvailable() {
    return available;
  }

  public abstract void playAction();

  public void setAvailable(boolean available) {
    this.available = available;
  }
```

```
  void setName(String name) {
    this.name = name;
  }

  public void setPlayer(Player player) {
    this.player = player;
  }

    public String toString() {
        return name;
    }
}
```

d. After

Cell.java

```
package edu.ncsu.monopoly;

public abstract class Cell {
  private boolean available = true;
  private String name;
  protected Player owner;

  public String getName() {
    return name;
  }

  public Player getPlayer() {
    return player;
  }

  public int getPrice() {
    return 0;
  }

  public boolean isAvailable() {
    return available;
  }

  public abstract void playAction();

  public void setAvailable(boolean available) {
    this.available = available;
  }

  void setName(String name) {
    this.name = name;
```

```
  }

  public void setPlayer(Player player) {
    this.player = player;
  }

    public String toString() {
        return name;
    }
}
```

e. Why refactor

Selain pada class Cell.java, terdapat juga beberapa class lain yang menggunakan field player ini, sehingga jika tidak dilakukan refactoring akan terjadi error. Dengan dilakukannya refactoring, semua class yang menggunakan field ini akan diganti dengan nama field yang baru dan bersesuaian.

f. How to refactor

Pada Eclipse :

1. Blok nama field yang akan di-rename secara bersamaan(refactoring)
2. Klik kanan pada mouse ataupun touchpad, lalu pilih refactor
3. Pilih rename dan ganti dengan nama field yang baru yang sesuai dengan peran dan fungsinya
4. Klik centang pada *Update Referenses*, *Update Textual Occurencens*, dan juga Rename getter dan setter nya
5. Klik Preview dan OK

2. Extract Local Variable

a. Problem

Keefektifan source code menjadi salah satu hal penting dalam implementasi program yang salah satunya adalah pemanggilan method yang sama menghasilkan hasil yang sama terlebih jika pemanggilannya lebih dari sekali, harus dihindari. Dalam hal ini adalah cell.getColorGroup()

b. Solution

Membuat variabel lokal terlebih dahulu dengan tujuan untuk menampung hasil dari method cell.getColorGroup() sehingga pemanggilan hanya dilakukan sekali

c. Before

GameBoard.addCell()

```
public void addCell(PropertyCell cell) {
  int propertyNumber = getPropertyNumberForColor(cell.getColorGroup());
  colorGroups.put(cell.getColorGroup(), new Integer(propertyNumber + 1));
      cells.add(cell);
}
```

d. After

GameBoard.addCell()

```
public void addCell(PropertyCell cell) {
    String colorGroup = cell.getColorGroup();
    int propertyNumber = getPropertyNumberForColor(cell.getColorGroup());
    colorGroups.put(cell.getColorGroup(), new Integer(propertyNumber + 1));
        cells.add(cell);
}
```

e. Why refactor

Pembuatan variabel lokal yang bertujuan untuk menampung hasil dari method yang dipanggil secara berlebihan(lebih dari sekali) dan menghasilkan hasil yang sama menjadi penting agar meningkatkan keefektifan program.

f. How to refactor

Pada Eclipse :

1. Blok method yang akan di-refactoring, lalu klik kanan pada mouse
2. Pilih Refactor dan Extract Local Variable
3. Masukkan nama variabel baru yang akan menampung hasil method yang bersangkutan
4. Preview dan OK

3. Extract Method

a. Problem

Terdapat kesamaan kode antar baris satu dengan baris lainnya, sehingga lebih baik dikelompokkan

b. Solution

Memindahkan kode ke method baru dan pada method yang lama, disertakan juga pemanggilan ke method yang baru dibentuk tersebut

c. Before

GameMaster.btnGetOutOfJailClicked()

```
public void btnGetOutOfJailClicked() {
    getCurrentPlayer().getOutOfJail();
    if(getCurrentPlayer().isBankrupt()) {
      gui.setBuyHouseEnabled(false);
      gui.setDrawCardEnabled(false);
      gui.setEndTurnEnabled(false);
      gui.setGetOutOfJailEnabled(false);
      gui.setPurchasePropertyEnabled(false);
      gui.setRollDiceEnabled(false);
      gui.setTradeEnabled(getCurrentPlayerIndex(),false);
    }
    else {
      gui.setRollDiceEnabled(true);
      gui.setBuyHouseEnabled(getCurrentPlayer().canBuyHouse());
      gui.setGetOutOfJailEnabled(getCurrentPlayer().isInJail());
    }
}
```

GameMaster.btnEndTurnClicked()

```
public void btnEndTurnClicked() {
    setAllButtonEnabled(false);
    getCurrentPlayer().getPosition().playAction();
    if(getCurrentPlayer().isBankrupt()) {
      gui.setBuyHouseEnabled(false);
      gui.setDrawCardEnabled(false);
      gui.setEndTurnEnabled(false);
      gui.setGetOutOfJailEnabled(false);
      gui.setPurchasePropertyEnabled(false);
      gui.setRollDiceEnabled(false);
      gui.setTradeEnabled(getCurrentPlayerIndex(),false);
      updateGUI();
    }
    else {
      switchTurn();
      updateGUI();
    }
}
```

PropertyCell.getRent()

```
public int getRent() {
    int rentToCharge = rent;
    String [] monopolies = player.getMonopolies();
    for(int i = 0; i < monopolies.length; i++) {
      if(monopolies[i].equals(colorGroup)) {
        rentToCharge = rent * 2;
      }
    }
    if(numHouses > 0) {
      rentToCharge = rent * (numHouses + 1);
    }
    return rentToCharge;
  }
```

d. After

GameMaster.setAllButtonDisabled()

```
private void setAllButtonDisabled() {
    gui.setBuyHouseEnabled(false);
    gui.setDrawCardEnabled(false);
    gui.setEndTurnEnabled(false);
    gui.setGetOutOfJailEnabled(false);
    gui.setPurchasePropertyEnabled(false);
    gui.setRollDiceEnabled(false);
```

```
    gui.setTradeEnabled(getCurrentPlayerIndex(), false);
}
```

GameMaster.btnGetOutOfJailClicked()

```
public void btnGetOutOfJailClicked() {
    getCurrentPlayer().getOutOfJail();
    if (getCurrentPlayer().isBankrupt()) {
        setAllButtonsDisabled();
    } else {
        gui.setRollDiceEnabled(true);
        gui.setBuyHouseEnabled(getCurrentPlayer().canBuyHouse());
        gui.setGetOutOfJailEnabled(getCurrentPlayer().isInJail());
    }
}
```

GameMaster.btnEndTurnClicked()

```
public void btnEndTurnClicked() {
    setAllButtonEnabled(false);
    getCurrentPlayer().getPosition().playAction();
    if (getCurrentPlayer().isBankrupt()) {
        setAllButtonsDisabled();
        updateGUI();
    } else {
        switchTurn();
        updateGUI();
    }
}
```

PropertyCell.rentForMonopolies()

```
private int rentForMonopolies(int rentToCharge) {
    String[] monopolies = owner.getMonopolies();
    for (int i = 0; i < monopolies.length; i++) {
        if (monopolies[i].equals(colorGroup)) {
            rentToCharge = rent * 2;
        }
    }
    return rentToCharge;
}
```

PropertyCell.getRent()

```
public int getRent() {
    int rentToCharge = rent;
    rentToCharge = rentForMonopolies(rentToCharge);
    if (numHouses > 0) {
        rentToCharge = rent * (numHouses + 1);
```

```
    }
    return rentToCharge;
}
```

e. Why refactor

Jika ada beberapa baris kode yang sama dan terulang pada class atau method lainnya, maka ada baiknya jika dijadikan satu method baru agar tidak terjadi pengulangan beberapa baris kode yang sama sehingga meningkatkan keefektifan program

f. How to refactor

Pada Eclipse :

1. Blok method yang akan di-refactoring(dijadikan method baru), lalu klik kanan pada mouse
2. Pilih Refactor dan Extract Method
3. Masukkan nama method baru yang akan menampung beberapa baris kode yang bersangkutan agar tidak terjadi pengulangan
4. Preview dan OK

4. Extract Subclass

a. Problem

Terdapat field pada beberapa class, tetapi field tersebut tidak digunakan seluruhnya, sehingga kurang efektif

b. Solution

Membuat subclass yang baru yang dijadikan sebagai parent dari 3 class yang terdapat field yang bersangkutan

c. Before

Cell.java()

```
package com.polban.tekpro.monopoly;
public abstract class Cell {
    private boolean available = true;
```

```
    private String name;
    protected Player owner;
    public String getName() {
        return name;
    }
    public Player getOwner() {
        return owner;
    }
    public int getPrice() {
        return 0;
    }
    public boolean isAvailable() {
        return available;
    }
    public abstract void playAction();
    public void setAvailable(boolean available) {
        this.available = available;
    }
    void setName(String name) {
        this.name = name;
    }
    public void setOwner(Player player) {
        this.owner = player;
    }

    public String toString() {
        return name;
    }
}
```

d. After

Cell.java()

```
package com.polban.tekpro.monopoly;
public abstract class Cell {
    private String name;
    public String getName() {
        return name;
    }
    public int getPrice() {
        return 0;
    }
    public abstract void playAction();
    void setName(String name) {
        this.name = name;
    }
    public String toString() {
        return name;
```

```
    }

    public boolean isAvailable() {
        return false;
    }
}
```

OwnedCell.java()

```
package com.polban.tekpro.monopoly;
public abstract class OwnedCell extends Cell {
    private boolean available = true;
    protected Player owner;
    public OwnedCell() {
        super();
    }
    public boolean isAvailable() {
        return available;
    }
    public void setAvailable(boolean available) {
        this.available = available;
    }
    public Player getOwner() {
        return owner;
    }
    public void setOwner(Player player) {
        this.owner = player;
    }
}
```

e. Why refactor

Terdapat method yang jarang digunakan yang sebaiknya method yang jarang digunakan tersebut dikumpulkan dan dipindahkan ke suatu subclass baru

f. How to refactor

Pada Eclipse :

1. Pilih salah satu class yang terdapat method yang jarang dipakai tersebut
2. Pilih Refactor dan Extract superclass
3. Masukkan nama class baru yang akan menampung method yang bersangkutan

4. Preview dan OK