LAPORAN TUGAS MINGGU 13
Praktikum Teknik Pemrograman
“Java Fundamental”

Laporan ini disusun untuk memenuhi Tugas Mata Kuliah Praktikum Teknik Pemrograman

Disusun oleh:

Falia Davina Gustaman 211524041

PROGRAM STUDI D4 TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN

INFORMATIKAPOLITEKNIK NEGERI BANDUNG

2022

A. Refactoring

1. Setup

Untuk melakukan proses refactoring dan testing menggunakan Junit ada beberapa hal yang harus dilakukan, yaitu :

- Membuat new java project lalu pastikan didalamnya sudah terapat folder src yang nantinya akan digunakan sebagai tempat kita menyimpan program
- Jika sudah memiliki project klik kanan pada folder src yang terdapat di dalam project yang kita buat sebelumnya lalu klik menu import → General → Archieve File
- Cari dan pilih file project yang akan digunakan
- Untuk melakukan testing menggunakan JUnit lakukan setup : klik kanan pada project → pilih menu Build Path → Add Library → JUnit
- Pilih versi JUnit yang akan digunakan
- Untuk melakukan running program, klik kanan pada project → run as → JUnit Test.

2. Rename Class Field

Pada task ini ditemukan masalah Class Cell field Bernama player tidak menjelaskan secara detail mengenai peran player didalam class Cell, sehingga kita perlu melakukan refactoring dengan mengganti nama fielnya dari player menjadi owner pada class Cell.

Source Code Class Cell sebelum dilakukan refactoring :

```
package com.polban.tekpro.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;
    }
}
```

Source Code Class Cell setelah dilakukan refactoring :

```
package com.polban.tekpro.monopoly;

public abstract class Cell {
        private String name;
        protected Player owner;

        public String getName() {
            return name;
        }

        public Player getOwner() {
            return owner;
        }

        public int getPrice() {
            return 0;
        }


        public abstract void playAction();

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

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

    public String toString() {
        return name;
    }

        public boolean isAvailable() {
            // TODO Auto-generated method stub
            return false;
        }
}
```

Mengapa harus dilakukan refactoring? Karena field player yang ada di dalam Class Cell telah dipakai oleh banyak class yang lain sehingga diperlukan proses refactoring untuk mengubah seluruh code yang didalamnya menggunakan field player ini termasuj method set dan get nya.

Langkah – langkah melakukan refactoring yaitu :

- Highlight field yang akan di refactor
- Klik kanan pada highlight lalu pilih menu refactor → rename
- Ganti nama field yang telah dihighlight tadi dengan nama field yang baru
- Klik centang pada pilihan : *"Update references", "Update Textual Occurances", "Rename getter"* dan *"Rename setter"*
- Klik preview untuk memastikan semua proses refactoring sudah tepat
- Jika sudah maka klik OK

3. Extract Method Variable

Pada task ini terdapat masalah yaitu kesamaan baris kode dengan beberapa baris code lainnya, sehingga kita perlu untuk memisahkan dan membuat method baru dan mengumpulkan proses yang sama.

Source Code GameMaster.java → GameMaster.btnGetOutOfJailClicked sebelum dilakukan refactoring :

```
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());
            }
    }
```

Source Code GameMaster.java → GameMaster.btnGetOutOfJailClicked setelah dilakukan refactoring :

```
public void btnGetOutOfJailClicked() {
            getCurrentPlayer().getOutOfJail();
            if (getCurrentPlayer().isBankrupt()) {
                  setAllButtonsDisabled();
            }
            else {
```

```
                    gui.setRollDiceEnabled(true);

        gui.setBuyHouseEnabled(getCurrentPlayer().canBuyHouse());

        gui.setGetOutOfJailEnabled(getCurrentPlayer().isInJail());
            }
    }
```

Source Code PropertyCell.java → PropertyCell.getRent() sebelum dilakukan refactoring :

```
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;
        }
```

Source Code PropertyCell.java → PropertyCell.getRent() setelah dilakukan refactoring :

```
public int getRent() {
            int rentToCharge = rent;
            String [] monopolies = owner.getMonopolies();
            rentToCharge = rentForMonopolies(rentToCharge, monopolies);
            if(numHouses > 0) {
                rentToCharge = rent * (numHouses + 1);
            }
            return rentToCharge;
        }
```

Source Code GameMaster.java → GameMaster.btnEndTurnClicked() sebelum dilakukan refactoring :

```
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();
```

```
    }
}
```

Source Code GameMaster.java → GameMaster.btnEndTurnClicked() setelah dilakukan refactoring :

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

Mengapa harus dilakukan refactoring? Karena suatu method lebih baik hanya mempunyai satu fungsi khusus agar kode dapat dengan mudah untuk dipakai kembali. Dan juga tujuan dari method tersebut tetap tefokus pada tujuan awal dibuatnya method tersebut.

Cara refactoring pada task ini yaitu :

- Pilih baris kode yang ingin dikelompokkan
- Klik kanan, pilih menu refactor → Extract Method
- Input nama method untuk hasil dari pengelompokkan tersebut
- Klik preview untuk memastikan perubahan yang terjadi
- Klik OK

4. Extract Local Variable

Pada task ini terdapat masalah yaitu pada class GameBoard method addCell, cell.getColorGroup() dipanggil sebanyak dua kali dan menghasilkan nilai yang sama. Proses pemanggilan method yang sama dengan hasil yang sama lebih dari satu kali akan menjadi kurang efektif karena proses akan berjalan lebih banyak, sehingga kita perlu membuat variabel local untuk menampung hasil dari cell.getColorGroup(), yang menjadikan method ini hanya dipanggil satu kali.

Source Code Cell.java setelah dilakukan refactoring :

```
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;
    }
}
```

Source Code Cell.java setelah dilakukan refactoring :

```
package com.polban.tekpro.monopoly;

public abstract class Cell {
        private String name;
        protected Player owner;

        public String getName() {
                return name;
        }

        public Player getOwner() {
                return owner;
        }

        public int getPrice() {
                return 0;
        }


        public abstract void playAction();

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

```
        public void setOwner Player player) {
                this.owner = player;
        }

    public String toString() {
         return name;
    }

        public boolean isAvailable() {
                // TODO Auto-generated method stub
                return false;
        }
}
```

Class 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;
        }

}
```

Mengapa perlu dilakukan refactor? Karena method yang jarang digunakan didalam superclass lebih baik dipindahkan ke class baru yang khusus menampung method yang jarang dipakai tersebut.

Cara melakukan refactor pada task ini, yaitu :

- Pilih salah satu subclass dari superclass yang menggunakan method yang jarang digunakan
- Klik kanan pada subclass tersebut, kemudian pilih menu refactor → Extract Superclass
- Input nama superclass yang baru
- Tambahkan class yang akan meng-extend class tersebut
- Klik Ok untuk membuat class baru
- Klik kanan pada superclass yang lama, pilih menu refactor → Push Down
- PIlih field beserta method setter dan getter nya untuk dipindahkan ke subclass yang telah dibuat
- Klik OK

B. Permasalahan yang dihadapi.

→ Saat mengerjakan task, laptop saya mengalami lag yang cukup berat sehingga menghambat saat pengerjaan tugas ini.

C. Solusi dari permasalahan yang dihadapi.

→ Sabar dan mencoba restart laptop kembali.