NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

I. Kasus (100%)

Analisa dan **lakukanlah refactoring** pada project java uas-code-reengineering ini. Adapun kriteria smell code yang harus dianalisa adalah:

1. **(LO 2 & LO 3, 20 poin) Object Oriented Design Smell**

Person.java (Sebelum Refactoring)

```
package uas.code.reengineering.satu;

class Person {
        public String code;
        public String name;
        public String address;
        public String phone;
        public String email;

        public Person(String code, String name, String address, String phone, String
email) {
                super();
                this.code = code;
                this.name = name;
                this.address = address;
                this.phone = phone;
                this.email = email;
        }

        public String getCode() {
                return code;
        }
        public void setCode(String code) {
                this.code = code;
        }

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

        public String getAddress() {
                return address;
        }
        public void setAddress(String address) {
                this.address = address;
        }

        public String getPhone() {
                return phone;
        }
        public void setPhone(String phone) {
                this.phone = phone;
        }

        public String getEmail() {
                return email;
        }
        public void setEmail(String email) {
                this.email = email;
        }
}
```

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Kode di atas melanggar salah satu Prinsip Desain dari Object Oriented, yaitu Don't Repeat Yourself (DRY) artinya jangan menulis kode yang sama berulang kali.

Contohnya Ketika mengerjakan proyek yang sama berkali-kali, seperti membuat profil perusahaan, buatlah satu template dasar, yang nanti bisa dikembangkan dan modifikasi sesuai kebutuhan klien daripada menulis ulang semuanya dari awal.

Analisa Refactoring:

Smell code :

Duplicate Code ditemukan dalam method constructor dan setter Class Person.

Penyebab :

Constructor Class Person update nilai atribut namun fungsi setter kelas ini sudah ada yang melakukan hal yang sama yang berarti ada Duplicate Code di Class ini.

Selain itu, jika ada tambahan validasi input ke Class ini sebelum memperbarui properties-nya, maka terjadi duplikasi logika di Constructor maupun setter dari Class Person.

Solusi :

Gunakan Method Setter di dalam Constructor untuk update semua fields nya.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Person.java (Setelah Refactoring)

```java
package solution.uas.code.reengineering.satu;

public class Person {
    public String code;
    public String name;
    public String address;
    public String phone;
    public String email;

    public Person(String code, String name, String address, String phone, String email) {
        /*
         * We can avoid all these if we use the setter methods inside the constructor to
update the properties.
         */
        super();
        setCode(code);
        setName(name);
        setAddress(address);
        setPhone(phone);
        setEmail(email);
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getName() {
        return name;
    }

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

2. **(LO 2 & LO 3, 20 poin) Abstraction Smell**

CourseData.java (Sebelum Refactoring)

```
package uas.code.reengineering.dua;

public class CourseData {
	String courseId;
	String courseName;
	String courseCredit;
}
```

Courses.java (Sebelum Refactoring)

```
package uas.code.reengineering.dua;

public class Courses {
	private CourseData couserData;

	public String getCourseId() {
		return couserData.courseId;
	}
	public void setCourseId(String courseId) {
		this.couserData.courseId = courseId;
	}

	public String getCourseName() {
		return couserData.courseName;
	}
	public void setCourseName(String courseName) {
		this.couserData.courseName = courseName;
	}

	public String getCourseCredit() {
		return couserData.courseCredit;
	}
	public void setCourseCredit(String courseCredit) {
		this.couserData.courseCredit = courseCredit;
	}
}
```

Kode di atas melanggar salah satu Prinsip Desain dari Object Oriented, yaitu Dependency Injection Principles (DIP) dimana memungkinkan programmer untuk menghapus dependensi dari Hard Coded sehingga aplikasi menjadi lebih rendah coupling nya.

Contohnya robot tidak boleh tergantung dengan satu alat saja, namun bisa diubah-ubah misalnya robot tidak boleh hanya tergantung pada spatula saja untuk memaskan, tapi bisa juga menggunakan pisau, penjepit atau centong dengan menambahkan sekrup yang bisa digunakan untuk bongkar pasang alat. Sekrup ini pada dunia programming seperti membuat Object baru agar memperkecil coupling yang terjadi.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Analisa Refactoring (Pendekatan Pertama):

Smell code :

Missing Abstraction dikarenakan pembuatan objek Class CourseData tidak ada di Class Courses sehingga bisa terjadi coupling yang besar karena atribut berasal dari objek yang dibuat yaitu class Courses.

Penyebab :

Objek Class CourseData tidak dibuat di Class Courses dan oleh karena itu, dengan menerapkan metode apa pun dari Class Kursus akan menghasilkan NullPointerException sebagai atribut courseData akan selalu tetap nol.

Solusi :

Dengan menggunakan dua Class terpisah dan dalam hal ini Constructor akan dideklarasikan di Class Courses yang akan membuat turunan dari Class CourseData dan menetapkan ke data di courseData.

CourseData.java (Setelah Refactoring)

```
package solution.pertama.uas.code.reengineering.dua;

public class CourseData {
        /*
         * This class represents a CourseData entity with the following attributes:
         * 1. courseId - represented by a string
         * 2. courseName - represented by a string
         * 3. courseCredit - represented by a string
 */
        String courseId;
        String courseName;
        String courseCredit;
}
```

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Courses.java (Setelah Refactoring)

```
package solution.pertama.uas.code.reengineering.dua;

public class Courses {
	private CourseData couserData;

	public Courses() {
		couserData = new CourseData();
	}

	public String getCourseId() {
		return couserData.courseId;
	}

	public void setCourseId(String courseId) {
		this.couserData.courseId = courseId;
	}

	public String getCourseName() {
		return couserData.courseName;
	}

	public void setCourseName(String courseName) {
		this.couserData.courseName = courseName;
	}

	public String getCourseCredit() {
		return couserData.courseCredit;
	}

	public void setCourseCredit(String courseCredit) {
		this.couserData.courseCredit = courseCredit;
	}
}
```

Kode Sebelum Refactoring juga melanggar salah satu Prinsip Desain dari Object Oriented, yaitu Single Responsibility Principle (SRP) dimana dimana sebuah Class diusahakan jangan sampai memiliki tugas yang diluar kemampuannya.

Contohnya di suatu kafe hanya ada satu pegawai saja sebagai kasir, satpam, koki, dan pelayan. Pada SRP ia harus dipisah sesuai dengan tanggung jawab masing-masing.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Analisa Refactoring (Pendekatan Kedua):

Smell code :

Missing Abstraction dikarenakan mendeklarasikan dua Class sehingga detail code tersebar dan terekspos di banyak tempat. Menyebabkan tanggungjawab di class yang bersangkutan menjadi tidak jelas (melanggar SRP).

Penyebab :

Ada penggunaan dua Class terpisah untuk mencapai abstraksi yang tidak direkomendasikan karena dalam kasus ini, belum benar-benar diperlukan dalam abstraksi untuk sebuah class.

Solusi :

Daripada menggunakan dua Class terpisah untuk mencapai abstraksi, hanya akan ada satu Class untuk menyimpan informasi tentang Courses dan di mana semua anggota data akan menjadi Private dan Accessors serta Mutators yang sesuai akan menjadi publik.

Courses.java (Setelah Refactoring)

```
package solution.kedua.uas.code.reengineering.dua;
public class Courses {
	private String coursed, courseName, courseCredit;

	public String getCourseId() {
		return courseId;
	}
	public void setCourseId(String courseId) {
		this.courseId = courseId;
	}
	public String getCourseName() {
		return courseName;
	}
	public void setCourseName(String courseName) {
		this.courseName = courseName;
	}
	public String getCourseCredit() {
		return courseCredit;
	}
	public void setCourseCredit(String courseCredit) {
		this.courseCredit = courseCredit;
	}
}
```

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

3. **(LO 2 & LO 3, 20 poin) Bad Code Smell Encapsulation**

Shape.java (Sebelum Refactoring)

```
package uas.code.reengineering.tiga;

public interface Shape {
	float ballVolume();
	float ballArea();
	float ballAround();
	float squareArea();
	float squareAround();
}
```

Bola.java (Sebelum Refactoring)

```
package uas.code.reengineering.tiga;

public class Bola implements Shape{
	private int side;

	public Bola(int side) {
		super();
		this.side = side;
	}

	@Override
	public float ballVolume() {
		// TODO Auto-generated method stub
		return (float) ((4/3)*Math.PI*this.side*this.side*this.side);
	}

	@Override
	public float ballArea() {
		// TODO Auto-generated method stub
		return (float) (4*Math.PI*this.side*this.side);
	}

	@Override
	public float ballAround() {
		// TODO Auto-generated method stub
		return (float) ((4/3)*Math.PI*this.side*this.side);
	}

	@Override
	public float squareArea() {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public float squareAround() {
		// TODO Auto-generated method stub
		return 0;
	}
}
```

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Square.java (Sebelum Refactoring)

```
package uas.code.reengineering.tiga;

public class Square implements Shape {
	private float side;

	public Square(float side) {
		super();
		this.side = side;
	}

	@Override
	public float ballVolume() {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public float ballArea() {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public float ballAround() {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public float squareArea() {
		// TODO Auto-generated method stub
		return this.side*this.side;
	}

	@Override
	public float squareAround() {
		// TODO Auto-generated method stub
		return 4*this.side;
	}
}
```

Kode di atas melanggar salah satu Prinsip Desain dari Object Oriented, yaitu Interface Segregation Principle (ISP) tidak memaksa method yang tidak dibutuhkan seperti menghitung volume bola, luas bola, dan luas permukaan bola di Class Square karena override dari Interface Shape.

Contohnya robot yang tidak punya sayap, seharusnya tidak diberikan tugas untuk terbang, tapi sesuaikan tugas dengan kebutuhan yaitu Berputar, Angkat Tangan, dan lain sebagainya.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Analisa Refactoring:

Smell code :

Leaky Encapsulation dikarenakan pemakaian interface yang terlalu lengkap sehingga membocorkan detail implementasi yang seharusnya tidak diperlukan dalam Interface Shape seperti penamaan setiap bangun seperti bola, persegi, dll.

Penyebab :

Class Square tidak membutuhkan Method ballVolume, ballArea dan ballAround dan Class Bola tidak memerlukan method squareArea() dan squareAround() merupakan contoh dari Refused Bequest.

Solusi :

Beri nama tanpa referensi ke subclass tertentu di method dalam interface Shape seperti Area() dan Around() dan Bola dan Square akan mengimplementasikan Interface Shape.

Polymorphism juga ikut berperan disini karena bergantung pada objek mana yang digunakan, method Area() dan Around() yang sesuai dengan Class bersangkutan yang akan dipanggil.

Volume() bukanlah method yang digunakan oleh kedua class tersebut dan oleh karena itu tidak boleh menjadi bagian dari interface sama sekali karena di kasus ini hanya diimplementasikan di Class Bola.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Shape.java (Setelah Refactoring)

```java
package solution.uas.code.reengineering.tiga;

public interface Shape {
	float area();
	float around();
}
```

Bola.java (Setelah Refactoring)

```java
package solution.uas.code.reengineering.tiga;

public class Bola implements Shape {
	private int side;

	public Bola(int side) {
		this.side = side;
	}
	//Not used Overriding
	public float volume() {
		return (float) ((4 / 3) * Math.PI * this.side * this.side * this.side);
	}

	@Override
	public float area() {
		return (float) (4 * Math.PI * this.side * this.side);
	}

	@Override
	public float around() {
		return (float) ((4 / 3) * Math.PI * this.side * this.side);
	}
}
```

Square.java (Setelah Refactoring)

```java
package solution.uas.code.reengineering.tiga;

public class Square implements Shape {
	private float side;

	public Square(float side) {
		this.side = side;
	}

	@Override
	public float area() {
		return this.side * this.side;
	}

	@Override
	public float around() {
		return 4 * this.side;
	}
}
```

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

4. **(LO 2 & LO 3, 20 poin) Modularization Smell**

CalculateShape.java (Sebelum Refactoring)

```
package uas.code.reengineering.empat;

public interface CalculateShape {
	int calculareAround();
	int calculareArea();
}
```

Triangle.java (Sebelum Refactoring)

```
package uas.code.reengineering.empat;

public abstract class Triangle {
	private int radius;
	public int getRadius() {
		return radius;
	}
	public void setRadius(int radius) {
		this.radius = radius;
	}
	public abstract int calculareAround();
	public int calculareArea() {
		return (22/7)*this.radius*this.radius;
	}
}
```

Square.java (Sebelum Refactoring)

```
package uas.code.reengineering.empat;

public class Square extends Triangle implements CalculateShape {
	private int side;
	public Square(int side) {
		super();
		this.side = side;
	}
	@Override
	public int calculareAround() {

		return 4*this.side;
	}
	@Override
	public int calculareArea() {
		return this.side*this.side;
	}
}
```

Kode di atas melanggar salah satu Prinsip Desain dari Object Oriented, yaitu Single Responsibility Principle (SRP) dimana sebuah Class diusahakan jangan sampai memiliki tugas yang diluar kemampuannya.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Contohnya di suatu kafe hanya ada satu pegawai saja sebagai kasir, satpam, koki, dan pelayan. Pada SRP ia harus dipisah sesuai dengan tanggung jawab masing-masing.

Analisa Refactoring:

Smell code :

Insufficient Modularization dimana ada logika yang salah untuk menghitung keliling dan menghitung luas untuk bentuk Segitiga karena dipaksa untuk dipakai di Class Square dan Tipe Data di Interface CalculateShape.

Penyebab :

Class segitiga tidak memiliki radius untuk area dan keliling karena digunakan di Class Persegi dari Interface CalculateShape sehingga Class segitiga secara tidak langsung dipaksa untuk memakai radius.

Solusi :

Hapus kata kunci abstract dan hapus metode abstrak di Class Triangle serta jangan extends Class Triangle di Class Square, hanya implementasikan interface CalculateShape saja di kelas Persegi.

Menerapkan method getter dan setter dan membuat variabel baru yaitu alas dan tinggi serta mengimplementasikan interface CalculateShape di Class Triangle dan Ubah Tipe Data di Interface CalculateShape menjadi double karena input tidak selalu bilangan bulat.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

CalculateShape.java (Setelah Refactoring)

```
package solution.uas.code.reengineering.empat;

public interface CalculateShape {
	double calculareAround();
	double calculareArea();
}
```

Triangle.java (Setelah Refactoring)

```
package solution.uas.code.reengineering.empat;
//Removed Abstract From Triangle Class
public class Triangle implements CalculateShape {
	private int base;
	private int height;

	public int getBase() {
		return base;
	}
	public void setBase(int base) {
		this.base = base;
	}

	public int getHeight() {
		return height;
	}
	public void setHeight(int height) {
		this.height = height;
	}

	@Override
	public double calculareAround(){
		return this.base + this.height + Math.sqrt((this.base*this.base) +
(this.height+this.height));
	}

	@Override
	public double calculareArea() {
		return 0.5*this.base*this.height;
	}
}
```

Square.java (Setelah Refactoring)

```
package solution.uas.code.reengineering.empat;
//Just Implement the Interface
public class Square implements CalculateShape {
	private int side;
	public Square(int side) {
		this.side = side;
	}

	@Override
	public double calculareAround() {
		return 4*this.side;
	}

	@Override
	public double calculareArea() {
		return this.side*this.side;
	}
}
```

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

5. **(LO 2 & LO 3, 20 poin) Hierarchy Smell**

MassMedia.java (Sebelum Refactoring)

```
package uas.code.reengineering.lima;

public class MassMedia {
	private int name;
	private int releaseDate;
	private String publisher;
	public MassMedia(int name, int releaseDate, String publisher) {
		super();
		this.name = name;
		this.releaseDate = releaseDate;
		this.publisher = publisher;
	}
	public void reading() {

	}
}
```

Megazine.java (Sebelum Refactoring)

```
package uas.code.reengineering.lima;

public class Megazine extends MassMedia {
	public Megazine(int name, int releaseDate, String publisher) {
		super(name, releaseDate, publisher);
		// TODO Auto-generated constructor stub
	}
	public void published() {}
	public void numberOfCopy() {}
}
```

Newspaper.java (Sebelum Refactoring)

```
package uas.code.reengineering.lima;

public class Newspaper extends MassMedia {
	public Newspaper(int name, int releaseDate, String publisher) {
		super(name, releaseDate, publisher);
		// TODO Auto-generated constructor stub
	}
	public void published() {}
	public void numberOfCopy() {}
}
```

Kode di atas melanggar salah satu Prinsip Desain dari Object Oriented, yaitu Liskov Substitution Principle (LSP) dimana terdapat SubClass memiliki tipe dan jumlah yang sama dengan fungsi yang berada pada class lainnya, maka bisa membuat abstract class dengan memanfaatkan fungsi extend.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Contohnya anak robot harus bisa melakukan apa yang ibunya bisa. Misalnya ibunya bisa buat teh, maka anaknya pun juga harus bisa melakukan hal yang sama.

Analisa Refactoring (Pendekatan Pertama):

Smell code :

Wide Hierarchy dimana banyaknya duplikasi method published() dan numberofcopy() terjadi pada Class Megazine dan Newspaper karena minimnya intermediate class sehingga tidak fleksibel dalam pengaplikasian hierarki.

Penyebab :

Class Megazine dan Newspaper sama sama memiliki Method dengan nama dan parameter yang sama sehingga duplikasi kode terjadi dikarenakan kurang paham pentingnya generalization dalam hierarki dan tidak mempedulikan adanya perpaduan behaviour yang bisa dipakai oleh subclass lainnya..

Solusi :

Buat class abstract baru bernama Marketting dikarenakan adanya kemiripan behaviour dari class Megazine dan Newspaper dengan menghadirkan abstract method published() dan numberOfCopy() untuk diaplikasikan pada Megazine dan Newspaper. Juga ubah Tipe Data untuk Name dan ReleaseDate menjadi String karena sesuai dengan peruntukkannya.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

MassMedia.java (Setelah Refactoring)

```java
package solution.pertama.uas.code.reengineering.lima;

public class MassMedia {
	private String name;
	private String releaseDate;
	private String publisher;
	public MassMedia(String name, String releaseDate, String publisher) {
		super();
		this.name = name;
		this.releaseDate = releaseDate;
		this.publisher = publisher;
	}
	public void reading() {

	}
}
```

Marketting.java (Setelah Refactoring)

```java
package solution.pertama.uas.code.reengineering.lima;

public abstract class Marketting {
	public abstract void published();
	public abstract void numberOfCopy();
}
```

Megazine.java (Setelah Refactoring)

```java
package solution.pertama.uas.code.reengineering.lima;

public class Megazine extends Marketting{
	public Megazine(String name, String releaseDate, String publisher) {
		super();
		// TODO Auto-generated constructor stub
	}

	@Override
	public void published() {
		// TODO Auto-generated method stub
	}

	@Override
	public void numberOfCopy() {
		// TODO Auto-generated method stub
	}
}
```

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Newspaper.java (Setelah Refactoring)

```
package solution.pertama.uas.code.reengineering.lima;

public class Newspaper extends Marketting{
	public Newspaper(String name, String releaseDate, String publisher) {
		super();
		// TODO Auto-generated constructor stub
	}

	@Override
	public void published() {
		// TODO Auto-generated method stub
	}

	@Override
	public void numberOfCopy() {
		// TODO Auto-generated method stub
	}
}
```

Kode Sebelum Refactoring juga melanggar Prinsip You Ain't Gonna Need It (YAGNI) dimana jika sesuatu method atau class tidak digunakan dalam waktu dekat, maka janganlah membuat spekulasi sendiri bahwa itu pasti akan digunakan.

Contohnya ada seseorang yang bekerja di sebuah startup yang menjual asuransi untuk bisnis perkapalan dimana sistem softwarenya dipecah menjadi dua komponen utama yaitu harga dan penjualan sehingga function untuk penjualan tidak dibuat dahulu sampai adanya penetapan harga jual yang relevan.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

Analisa Refactoring (Pendekatan Kedua):

Smell code :

Speculative Hierarchy dimana terdapat class yang dibuat untuk keperluan sewaktu-waktu fitur tersebut akan ada di waktu yang akan mendatang atas imajinasi developer sendiri.

Penyebab :

Terlalu prediktif dengan pembuatan software dimana Class Megazine dan Newspaper sama sama memiliki Method dengan nama dan parameter yang sama tapi disini fokus kepada pendataan barangnya terlebih dahulu dilihat dari constructor di MassMedia dan sekarang bukan untuk membuat function published() dan numberOfCopy().

Solusi :

Hapus method published() dan numberOfCopy() di Class Megazine dan Newspaper sehingga hal-hal yang bersifat spekulatif bisa dihilangkan sehingga apa yang ada di kode hanya fokus utama untuk saat ini saja dan memudahkan proses dokumentasi software agar tidak terjadinya kesalahpahaman kedepannya.

NAME : Fiona Varencia Tendio **Course : Code Reengineering**
NIM : 2301905543 **Course Code : COMP6106**
CLASS : LK-01 **Faculty / Department : School of Computer Science**

MassMedia.java (Setelah Refactoring)

```java
package solution.kedua.uas.code.reengineering.lima;

public class MassMedia {
	private String name;
	private String releaseDate;
	private String publisher;
	public MassMedia(String name, String releaseDate, String publisher) {
		super();
		this.name = name;
		this.releaseDate = releaseDate;
		this.publisher = publisher;
	}
	public void reading() {

	}
}
```

Megazine.java (Setelah Refactoring)

```java
package solution.kedua.uas.code.reengineering.lima;

public class Megazine extends MassMedia {
	public Megazine(String name, String releaseDate, String publisher) {
		super(name, releaseDate, publisher);
		// TODO Auto-generated constructor stub
	}
}
```

Newspaper.java (Setelah Refactoring)

```java
package solution.kedua.uas.code.reengineering.lima;

public class Newspaper extends MassMedia {
	public Newspaper(String name, String releaseDate, String publisher) {
		super(name, releaseDate, publisher);
		// TODO Auto-generated constructor stub
	}
}
```