LAPORAN PRAKTIKUM
Inheritance, Abstract Class and Interface

Disusun Oleh:
Dzakir Tsabit Asy Syafiq (241511071)
Jurusan Teknik Komputer dan Informatika

Program Studi D-3 Teknik Informatika
Politeknik Negeri Bandung
12/09/2025

TASK 1: The Circle and Cylinder Classes

Task 1.1: Modify class Circle

Kode Circle.java (Modified):

```java
/**
 * The Circle class models a circle with a radius and color.
 */
public class Circle { // Save as "Circle.java"
    // private instance variable, not accessible from outside this class

    private double radius;
    private String color;
    // Constructors (overloaded)

    /**
     * Constructs a Circle instance with default value for radius and color
     */
    public Circle() { // 1st (default) constructor
        radius = 1.0;
        color = "red";
    }

    /**
     * Constructs a Circle instance with the given radius and default color
     */
    public Circle(double r) { // 2nd constructor
        radius = r;
        color = "red";
    }
    // constructor with given radius and color
    public Circle(double radius, String color) { // 3rd constructor
        this.radius = radius;
        this.color = color;
    }
    // retunr color
    public String getColor() {
        return color;
    }
    /**
     * Returns the radius
     */
    public double getRadius() {
        return radius;
    }

    /**
     * Returns the area of this Circle instance
     */
    public double getArea() {
        return radius * radius * Math.PI;
    }
    //setter
    public void setColor(String color) {
        this.color = color;
    }
    public void setRadius(double radius) {
        this.radius = radius;
    }

    /**
     * Return a self-descriptive string of this instance in the form of
     * Circle[radius=?,color=?]
     */

    @Override
    public String toString() {
        return "Circle[radius=" + radius + " color=" + color + "]";
    }
}
```

1.2: *Overriding* method getArea() di Cylinder

Kode Cylinder.java (Modified):

```
public class Cylinder extends Circle { // Save as "Cylinder.java"

    private double height; // private variable

    // Constructor with default color, radius and height
    public Cylinder()

    {
        super(); // call superclass no-arg constructor Circle()
        height = 1.0;
    }
    // Constructor with default radius, color but given height

    public Cylinder(double height) {
        super(); // call superclass no-arg constructor Circle()
        this.height = height;
    }
    // Constructor with default color, but given radius, height

    public Cylinder(double radius, double height) {
        super(radius); // call superclass constructor Circle(r)
        this.height = height;
    }

    // A public method for retrieving the height
    public double getHeight() {
        return height;
    }

    // Override the getArea() method of Circle
    // to return the surface area of this cylinder
    @Override
    public double getArea() {
        // Luas permukaan silinder: 2πrh + 2πr^2
        return 2 * Math.PI * getRadius() * height + 2 * Math.PI * Math.pow(getRadius(), 2);
    }

    // A public method for computing the volume of cylinder
    // use superclass method getArea() to get the base area
    public double getVolume() {
        return getArea() * height;
    }
    @Override
    public String toString() {
        return "Cylinder: subclass of " + super.toString() + " height=" + height;
    }
}
```

1.3: Provide a toString() method di Cylinder

Sudah diimplementasikan di atas (lihat method toString() di class Cylinder). Method ini memanggil super.toString() untuk mendapatkan representasi string dari Circle, lalu menambahkan informasi tinggi silinder.

1.4: Buat class TestCylinder

Kode *TestCylinder.java* :

```
// TestCylinder.java
public class TestCylinder {
    public static void main(String[] args) {
        Cylinder c1 = new Cylinder();
        System.out.println("Cylinder:"
                           + " radius=" + c1.getRadius()
                           + " height=" + c1.getHeight()
                           + " base area=" + c1.getArea()
                           + " volume=" + c1.getVolume());

        Cylinder c2 = new Cylinder(10.0);
        System.out.println("Cylinder:"
                           + " radius=" + c2.getRadius()
                           + " height=" + c2.getHeight()
                           + " base area=" + c2.getArea()
                           + " volume=" + c2.getVolume());

        Cylinder c3 = new Cylinder(2.0, 10.0);
        System.out.println("Cylinder:"

                         + " radius=" + c3.getRadius()

                         + " height=" + c3.getHeight()

                         + " base area=" + c3.getArea()

                         + " volume=" + c3.getVolume());


        System.out.println(c1.toString());

        System.out.println(c2.toString());

        System.out.println(c3.toString());

    }

}
```

Output dari program TestCylinder akan menunjukkan:

- Nilai radius, tinggi, luas alas, dan volume untuk setiap instance Cylinder yang dibuat.
- Perbedaan dalam nilai-nilai ini berdasarkan konstruktor yang digunakan (nilai default atau nilai yang diberikan).
- Luas permukaan yang benar (karena getArea() sudah di-*override*).
- Representasi string dari setiap objek Cylinder, menunjukkan bahwa metode toString() berfungsi dan memanggil toString() dari superclass Circle.

Contoh output:

```
Cylinder: radius=1.0 height=1.0 base area=12.566370614359172 volume=12.566370614359172

Cylinder: radius=1.0 height=10.0 base area=69.11503837897544 volume=691.1503837897544

Cylinder: radius=2.0 height=10.0 base area=150.79644737231007 volume=1507.9644737231006

Cylinder: subclass of Circle[radius=1.0 color=red] height=1.0

Cylinder: subclass of Circle[radius=1.0 color=red] height=10.0

Cylinder: subclass of Circle[radius=2.0 color=red] height=10.0
```

Analisis Hubungan

- Inheritance: Cylinder *mewarisi* properti radius dan color dari Circle. Ini berarti bahwa setiap objek Cylinder secara otomatis memiliki properti ini.
- Override: Metode getArea() di-*override* di Cylinder untuk memberikan implementasi yang spesifik untuk silinder (luas permukaan), bukan lingkaran.
- Super:
  - Konstruktor Cylinder menggunakan super() untuk memanggil konstruktor Circle dan menginisialisasi radius dan color. Ini memastikan bahwa bagian Circle dari objek Cylinder diinisialisasi dengan benar.
  - Metode toString() di Cylinder menggunakan super.toString() untuk memanggil metode toString() dari Circle, memungkinkan Cylinder untuk

menggunakan kembali representasi string dari lingkaran dan menambahkan informasi spesifik silinder.

Kesimpulan:

Dengan mengikuti langkah-langkah ini, Anda seharusnya dapat menyelesaikan *task* 1.1 - 1.6. Ingatlah untuk menguji kode Anda secara menyeluruh dan memberikan penjelasan yang jelas dan ringkas dalam laporan Anda.

TASK 2: Superclass Shape and Subclasses

Task 2.1: Shape, Circle, Rectangle, dan Square Classes

1. **Shape Class:**
    - Dua instance variable: color (String) dan filled (boolean).
    - Dua constructors:
        - No-arg constructor: Initializes color ke "green" dan filled ke true.
        - Constructor dengan parameter color dan filled.
    - Getter dan setter untuk semua instance variables (termasuk isFilled() untuk filled).
    - Method toString(): Mengembalikan string dengan format "A Shape with color of xxx and filled/Not filled".
2. **Circle Class (Subclass of Shape):**
    - Instance variable: radius (double).
    - Tiga constructors:
        - No-arg constructor: Initializes radius ke 1.0.
        - Constructor dengan parameter radius.
        - Constructor dengan parameter radius, color, dan filled.
    - Getter dan setter untuk radius.
    - Methods getArea() dan getPerimeter().
    - Override method toString(): Mengembalikan string dengan format "A Circle with radius=xxx, which is a subclass of yyy", dimana yyy adalah output dari toString() method dari superclass.
3. **Rectangle Class (Subclass of Shape):**
    - Instance variables: width (double) dan length (double).

    - Tiga constructors:
        - No-arg constructor: Initializes width dan length ke 1.0.
        - Constructor dengan parameter width dan length.
        - Constructor dengan parameter width, length, color, dan filled.
    - Getter dan setter untuk width dan length.
    - Methods getArea() dan getPerimeter().
    - Override method toString(): Mengembalikan string dengan format "A Rectangle with width=xxx and length=zzz, which is a subclass of yyy", dimana yyy adalah output dari toString() method dari superclass.

4. **Square Class (Subclass of Rectangle):**
    - Tidak ada instance variables tambahan
      (menggunakan width dan length dari Rectangle).
    - Constructors (panggil superclass constructors dengan nilai yang sesuai).
    - Override setLength() dan setWidth() untuk memastikan
      bahwa width dan length selalu sama (untuk menjaga properti persegi).
    - Override method toString(): Mengembalikan string dengan format "A Square with side=xxx, which is a subclass of yyy", dimana yyy adalah output
      dari toString() method dari superclass.
    - Implementasi getArea dan getPerimeter() tidak perlu di override karena sudah benar di class Rectangle.

Kode Shape.java :

```java
public class Shape {

    private String color;
    private boolean filled;

    public Shape() {
        this.color = "green";
        this.filled = true;
    }

    public Shape(String color, boolean filled) {
        this.color = color;
        this.filled = filled;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public boolean isFilled() {
        return filled;
    }

    public void setFilled(boolean filled) {
        this.filled = filled;
    }

    @Override
    public String toString() {
        return "A Shape with color of " + color + " and " + (filled ? "filled" : "Not filled");
    }
}
```

Kode Circle.java :

```
public class Circle extends Shape {

    private double radius;

    public Circle() {
        super();
        this.radius = 1.0;
    }

    public Circle(double radius) {
        super();

        this.radius = radius;

    }


    public Circle(double radius, String color, boolean filled) {

        super(color, filled);

        this.radius = radius;

    }


    public double getRadius() {

        return radius;

    }


    public void setRadius(double radius) {

        this.radius = radius;

    }


    public double getArea() {

        return Math.PI * radius * radius;

    }


    public double getPerimeter() {

        return 2 * Math.PI * radius;

    }


    @Override

    public String toString() {

        return "A Circle with radius=" + radius + ", which is a subclass of " +
super.toString();

    }

}
```

Kode Rectangle.java :

```
public class Rectangle extends Shape {


    private double width;
    private double length;

    public Rectangle() {
        super();
        this.width = 1.0;
        this.length = 1.0;
    }

    public Rectangle(double width, double length) {
        super();
        this.width = width;
        this.length = length;
    }

    public Rectangle(double width, double length, String color, boolean filled) {
        super(color, filled);
        this.width = width;
        this.length = length;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }

    public double getArea() {
        return width * length;
    }
```

```
    public double getPerimeter() {

        return 2 * (width + length);

    }


    @Override

    public String toString() {

        return "A Rectangle with width=" + width + " and length=" + length + ", which is a subclass of " + super.toString();

    }

}
```

Kode Square.java :

```
public class Square extends Rectangle {


    public Square() {

        super(); // width dan length = 1.0

    }


    public Square(double side) {

        super(side, side);

    }


    public Square(double side, String color, boolean filled) {

        super(side, side, color, filled);

    }


    public double getSide() {

        return getWidth(); // atau getLength(), karena sama

    }


    public void setSide(double side) {

        setWidth(side);

        setLength(side);

    }


    @Override
    public void setWidth(double side) {

        super.setWidth(side);

        super.setLength(side);

    }


    @Override

    public void setLength(double side) {

        super.setWidth(side);

        super.setLength(side);

    }


    @Override

    public String toString() {

        return "A Square with side=" + getSide() + ", which is a subclass of " +
super.toString();

    }

}
```

Kode TestShape.java (main) :

```
public class TestShape {


    public static void main(String[] args) {

        Shape s1 = new Shape("red", false);

        System.out.println(s1);


        Circle c1 = new Circle(5.5, "blue", true);

        System.out.println(c1);

        System.out.println("Area: " + c1.getArea());

        System.out.println("Perimeter: " + c1.getPerimeter());


        Rectangle r1 = new Rectangle(1.2, 3.4, "orange", false);

        System.out.println(r1);

        System.out.println("Area: " + r1.getArea());

        System.out.println("Perimeter: " + r1.getPerimeter());


        Square sq1 = new Square(6.6, "yellow", true);

        System.out.println(sq1);

        System.out.println("Area: " + sq1.getArea());
        System.out.println("Perimeter: " + sq1.getPerimeter());


        sq1.setSide(10);

        System.out.println(sq1);

        System.out.println("Area: " + sq1.getArea());

        System.out.println("Perimeter: " + sq1.getPerimeter());

    }

}
```

Output Program :

```
A Shape with color of red and Not filled

A Circle with radius=5.5, which is a subclass of A Shape with color of blue and filled

Area: 95.03317777109123

Perimeter: 34.55751918948772

A Rectangle with width=1.2 and length=3.4, which is a subclass of A Shape with color of orange
and Not filled

Area: 4.08

Perimeter: 9.2

A Square with side=6.6, which is a subclass of A Rectangle with width=6.6 and length=6.6, which
is a subclass of A Shape with color of yellow and filled

Area: 43.559999999999995

Perimeter: 26.4

A Square with side=10.0, which is a subclass of A Rectangle with width=10.0 and length=10.0,
which is a subclass of A Shape with color of yellow and filled

Area: 100.0

Perimeter: 40.0
```

Analisis Output:

- **Shape s1 = new Shape("red", false);**
  - System.out.println(s1); -> A Shape with color of red and Not filled
  - Ini menunjukkan bahwa konstruktor Shape dan method toString() berfungsi dengan benar.
- **Circle c1 = new Circle(5.5, "blue", true);**
  - System.out.println(c1); -> A Circle with radius=5.5, which is a subclass of A Shape with color of blue and filled
  - Ini menunjukkan bahwa konstruktor Circle dengan parameter dan *overriding* method toString() berfungsi dengan benar. Perhatikan

    - bagaimana ia menggunakan super.toString() untuk memasukkan representasi Shape.
    - System.out.println("Area: " + c1.getArea()); -> Area: 95.03317777109123
    - Ini menunjukkan bahwa method getArea() di Circle dihitung dengan benar.
    - System.out.println("Perimeter: " + c1.getPerimeter()); -> Perimeter: 34.55751918948772
    - Ini menunjukkan bahwa method getPerimeter() di Circle dihitung dengan benar.
- **Rectangle r1 = new Rectangle(1.2, 3.4, "orange", false);**
    - System.out.println(r1); -> A Rectangle with width=1.2 and length=3.4, which is a subclass of A Shape with color of orange and Not filled
    - Ini menunjukkan bahwa konstruktor Rectangle dengan parameter dan *overriding* method toString() berfungsi dengan benar. Perhatikan bagaimana ia menggunakan super.toString() untuk memasukkan representasi Shape.
    - System.out.println("Area: " + r1.getArea()); -> Area: 4.08
    - Ini menunjukkan bahwa method getArea() di Rectangle dihitung dengan benar.
    - System.out.println("Perimeter: " + r1.getPerimeter()); -> Perimeter: 9.2
    - Ini menunjukkan bahwa method getPerimeter() di Rectangle dihitung dengan benar.
- **Square sq1 = new Square(6.6, "yellow", true);**
    - System.out.println(sq1); -> A Square with side=6.6, which is a subclass of A Rectangle with width=6.6 and length=6.6, which is a subclass of A Shape with color of yellow and filled
    - Ini menunjukkan bahwa konstruktor Square dengan parameter dan *overriding* method toString() berfungsi dengan benar. Perhatikan bagaimana ia menggunakan super.toString() untuk memasukkan representasi Rectangle dan Shape.
    - System.out.println("Area: " + sq1.getArea()); -> Area: 43.559999999999995
    - Ini menunjukkan bahwa method getArea() di Square (yang diwarisi dari Rectangle) dihitung dengan benar.
    - System.out.println("Perimeter: " + sq1.getPerimeter()); -> Perimeter: 26.4
    - Ini menunjukkan bahwa method getPerimeter() di Square (yang diwarisi dari Rectangle) dihitung dengan benar.
- **sq1.setSide(10);**

- System.out.println(sq1); -> A Square with side=10.0, which is a subclass of A Rectangle with width=10.0 and length=10.0, which is a subclass of A Shape with color of yellow and filled
- Ini menunjukkan bahwa
  method setSide() (yang *override* setWidth() dan setLength()) berfungsi dengan benar, mengubah kedua sisi persegi menjadi 10.0.
- System.out.println("Area: " + sq1.getArea()); -> Area: 100.0
- Ini menunjukkan bahwa method getArea() setelah perubahan sisi dihitung dengan benar.
- System.out.println("Perimeter: " + sq1.getPerimeter()); -> Perimeter: 40.0
- Ini menunjukkan bahwa method getPerimeter() setelah perubahan sisi dihitung dengan benar.

TASK 3: Multiple Inheritance (Abstract Class)

Task 3.1: Extending the Sortable abstract class

Kode Abstrack Sortable.java (compare()) :

```
abstract class Sortable {

    public abstract int compare(Sortable b);

    public static void shell_sort(Sortable[] a) {
        int n = a.length;

        // Start with a big gap, then reduce the gap
        for (int gap = n / 2; gap > 0; gap /= 2) {
            // Do a gapped insertion sort for this gap size
            for (int i = gap; i < n; i++) {
                Sortable temp = a[i];
                int j;

                // Shift earlier gap-sorted elements up until the correct location for a[i] is
found
                for (j = i; j >= gap && a[j - gap].compare(temp) > 0; j -= gap) {
                    a[j] = a[j - gap];
                }
```

```java
                // Put temp (the original a[i]) in its correct location
                a[j] = temp;
            }
        }
    }
}
```

Kode Employee.java:

```java
class Employee extends Sortable {

    private String name;
    private double salary;
    private int hireday;
    private int hiremonth;
    private int hireyear;

    public Employee(String n, double s, int day, int month, int year) {
        name = n;
        salary = s;
        hireday = day;
        hiremonth = month;
        hireyear = year;
    }

    public void print() {
        System.out.println(name + " " + salary + " " + hireYear());
    }

    public void raiseSalary(double byPercent) {
        salary *= 1 + byPercent / 100;
    }

    public int hireYear() {
        return hireyear;
    }
    // Implement abstract method from Sortable
    @Override
    public int compare(Sortable b) {
        Employee eb = (Employee) b;
        if (salary < eb.salary) {
            return -1;
        }
        if (salary > eb.salary) {
            return +1;
        }
        return 0;
    }

    // Getters for access from Manager
    public double getSalary() {
        return salary;
    }

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

Kode EmployeeTest.java:

```
public class EmployeeTest {

    public static void main(String[] args) {
        Employee[] staff = new Employee[3];
        staff[0] = new Employee("Antonio Rossi", 2000000, 1, 10, 1989);
        staff[1] = new Employee("Maria Bianchi", 2500000, 1, 12, 1991);
        staff[2] = new Employee("Isabel Vidal", 3000000, 1, 11, 1993);
        int i;
        for (i = 0; i < 3; i++) {
            staff[i].raiseSalary(5);
        }
        for (i = 0; i < 3; i++) {
            staff[i].print();
        }
    }
}
```

Kode Manager.java:

```
import java.util.Calendar;
import java.util.GregorianCalendar;

class Manager extends Employee {

    private String secretaryName;

    public Manager(String n, double s, int d, int m, int y) {
        super(n, s, d, m, y);
        secretaryName = "";
    }

    @Override
    public void raiseSalary(double byPercent) {
        // add 1/2% bonus for every year of service
        GregorianCalendar todaysDate = new GregorianCalendar();
        int currentYear = todaysDate.get(Calendar.YEAR);
        double bonus = 0.5 * (currentYear - hireYear());
        super.raiseSalary(byPercent + bonus);
    }

    public String getSecretaryName() {
        return secretaryName;
    }

    public void setSecretaryName(String secretaryName) {
        this.secretaryName = secretaryName;
    }

    // Manager inherits compare method from Employee
}
```

Kode ManagerTest.java:

```
public class ManagerTest {

    public static void main(String[] args) {
        Employee[] staff = new Employee[3];
        staff[0] = new Employee("Antonio Rossi", 2000000, 1, 10, 1989);
        staff[1] = new Manager("Maria Bianchi", 2500000, 1, 12, 1991);
        staff[2] = new Employee("Isabel Vidal", 3000000, 1, 11, 1993);
        int i;
        for (i = 0; i < 3; i++) {
            staff[i].raiseSalary(5);
        }
        for (i = 0; i < 3; i++) {
            staff[i].print();
        }
    }
}
```

Output Task 3.1:

```
Antonio Rossi 2100000.0 1989
Maria Bianchi 2625000.0 1991
Isabel Vidal 3150000.0 1993
```

Penjelasan Task 3.1:

Case 1 - Employee extends Sortable:

- Employee class berhasil meng-extend abstract class Sortable
- Implement method compare() untuk membandingkan salary
- Shell sort berfungsi dengan baik untuk mengurutkan Employee berdasarkan salary

Case 2 - Multiple Inheritance Problem:

class Manager extends Employee extends Sortable // TIDAK BISA!

Masalah: Java tidak mendukung multiple inheritance untuk class. Manager tidak bisa extends Employee dan Sortable sekaligus.

Solusi:

1. Manager extends Employee, dan Employee extends Sortable
2. Manager secara otomatis mewarisi semua method dari Sortable melalui Employee
3. Manager bisa menggunakan method compare() yang diimplementasikan di Employee
4. Jika perlu behavior compare yang berbeda, Manager bisa override method compare()

Alternatif Solusi dengan Interface:

```
interface Comparable {
    int compare(Comparable b);
}

class Employee implements Comparable {
    // ... implementation
}

class Manager extends Employee {
    // Automatically implements Comparable through Employee
    // Can override compare() if needed
}
```

Kesimpulan :

Konsep Inheritance yang dipelajari:

1. **Basic Inheritance**: Class child mewarisi properties dan methods dari parent class
2. **Constructor Chaining**: Menggunakan super() untuk memanggil constructor parent
3. **Method Overriding**: Child class dapat override method parent dengan @Override
4. **Super Keyword**: Mengakses method parent class dengan super.methodName()
5. **Abstract Classes**: Class yang tidak bisa diinstantiasi, harus di-extend
6. **Multiple Inheritance**: Java tidak mendukung multiple inheritance untuk class, tapi bisa diselesaikan dengan interface atau hierarchy yang tepat

Keuntungan Inheritance:

- Code reusability
- Easier maintenance
- Polymorphism support

- Logical hierarchy representation

Best Practices:

- Gunakan @Override annotation
- Panggil super() di constructor child class
- Override toString() untuk debugging yang lebih baik
- Implementasikan abstract methods di concrete classes