Mats Vik Pettersen
Programering 2

Oblig 3

1.1

Polymorphism (Polymorfisme)

Polymorfisme betyr "mange former" og refererer til muligheten for at en metode eller et objekt kan ha flere former. Det finnes to hovedtyper av polymorfisme i Java:

1. **Kjøretidspolymorfisme (Runtime Polymorphism)** – Bruk av **metodeoverstyring (method overriding)**, der en subklasse gir en spesifikk implementasjon av en metode som allerede er definert i en superklasse.
2. **Kompileringstidspolymorfisme (Compile-time Polymorphism)** – Bruk av **metodeoverlasting (method overloading)**, der flere metoder har samme navn, men forskjellige parameterlister.

```java
class Animal {
    void makeSound() {
        System.out.println(x:"Animal makes a sound");
    }
}

class Dog extends Animal {
    void makeSound() {  // Overstyring (runtime polymorphism)
        System.out.println(x:"Dog barks");
    }
}

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Animal myDog = new Dog();
        myDog.makeSound();  // Kaller Dog sin metode, ikke Animal sin
    }
}
```

Dette kalles **dynamisk binding**, fordi metoden som skal brukes bestemmes ved kjøretid.

Mats Vik Pettersen
Programering 2

Refaktorere

Refaktorering betyr å forbedre kode uten å endre dens funksjonalitet. Målet er å gjøre koden mer lesbar, vedlikeholdbar og effektiv.

Eksempler på refaktorering:

- Fjerne duplisert kode
- Dele opp lange metoder i mindre metoder
- Gi variabler og metoder mer beskrivende navn
- Bruke designmønstre for å forbedre struktur

Eksempel på refaktorering:

```java
// Før refaktorering
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
}

// Etter refaktorering (for bedre utvidbarhet)
public class Calculator {
    public int add(int... numbers) {
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        return sum;
    }
}
```

Static (metode, variabel)

Static betyr at en metode eller variabel tilhører **klassen** i stedet for et objekt av klassen. Det betyr at man kan bruke metoden eller variabelen uten å lage et objekt av klassen.

1. **Static variabel (klassevariabel)** – Deler samme verdi på tvers av alle instanser av klassen.

```java
class Example {
    static int count = 0;  // Denne variabelen deles av alle objektene
}
```

Static metode – Kan kalles uten å lage et objekt.

```java
class Utility {
    static void printMessage() {
        System.out.println(x:"Hello from a static method!");
    }
}

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Utility.printMessage(); // Kan kalles direkte
    }
}
```

Begrensning: En statisk metode kan ikke få tilgang til ikke-statiske variabler direkte.

Final (klasse, metode, variabel)

Final brukes for å gjøre noe **konstant** eller forhindre endringer.

1. **Final variabel** – Verdien kan ikke endres etter at den er satt.

```java
class Example {
    final int MAX_VALUE = 100;
}
```

Final metode – Kan ikke overstyres av subklasser.

```java
class Parent {
    final void show() {
        System.out.println(x:"This cannot be overridden");
    }
}
```

Final klasse – Kan ikke arves av andre klasser.

```java
final class CannotBeExtended {
    void display() {
        System.out.println(x:"This class cannot be extended");
    }
}
```

Abstract (klasse, metode)

Abstract brukes for å lage en mal for subklasser.

1. **Abstrakt klasse** – Kan ikke instansieres direkte og kan ha både abstrakte og vanlige metoder.
2. **Abstrakt metode** – Har ingen implementasjon i superklassen, men må implementeres i subklasser.

Eksempel:

```java
abstract class Animal {
    abstract void makeSound();  // Må implementeres av subklasser
}

class Dog extends Animal {
    void makeSound() {
        System.out.println(x:"Bark!");
    }
}
```

Fordel: Lar oss definere en felles struktur for alle subklasser.

Interface *(utsettes til neste oblig, men her er en forklaring)*

Et interface er en kontrakt som klasser kan implementere. Det inneholder **kun abstrakte metoder** (før Java 8) og brukes ofte for å oppnå **flermultiple arv** i Java.

Eksempel:

Mats Vik Pettersen
Programering 2

```java
interface Animal {
    void makeSound();
}

class Cat implements Animal {
    public void makeSound() {
        System.out.println(x:"Meow!");
    }
}
```

Mats Vik Pettersen
Programering 2

1.2

```
/*classDiagram
    class Episode {
        - String title
        - int episodeNumber
        - int seasonNumber
        - int runtime
        + Episode(title, episodeNumber, seasonNumber, runtime)
        + Episode(title, episodeNumber, seasonNumber)
        + getTitle(): String
        + setTitle(title: String)
        + getEpisodeNumber(): int
        + setEpisodeNumber(episodeNumber: int)
        + getSeasonNumber(): int
        + setSeasonNumber(seasonNumber: int)
        + getRuntime(): int
        + setRuntime(runtime: int)
        + toString(): String
    }

    class TVSeries {
        - String title
        - String description
        - LocalDate releaseDate
        - ArrayList<Episode> episodes
        - double averageRunTime
        - int numSeasons
        + TVSeries(title: String, description: String, releaseDate: LocalDate)
        + getTitle(): String
        + setTitle(title: String)
        + getDescription(): String
        + setDescription(description: String)
        + getReleaseDate(): LocalDate
        + setReleaseDate(releaseDate: LocalDate)
        + getAverageRunTime(): double
        + getNumSeasons(): int
        + addEpisode(episode: Episode)
        + getEpisodesInSeason(season: int): ArrayList<Episode>
        + toString(): String
    }

    class Main {
        + main(args: String[]): void
    }

    TVSeries "1" -- "*" Episode : inneholder
    Main --> TVSeries : bruker
    Main --> Episode : bruker*/
```