Løsningsforslag eksamen V21

INF101 - 9/6 2021

Typeparameter

- Rett alternativ: Alle datatyper som er subtyper av ShoppingItem
- Dette er en oppgave om generics
- Det er i definisjonen av klassen parameteren T defineres

```
public class Grid<T extends ShoppingItem> {
```

- T extends ShoppingItem betyr at T kan enten være ShoppingItem eller en hvilken som helst klasse som arver fra ShoppingItem

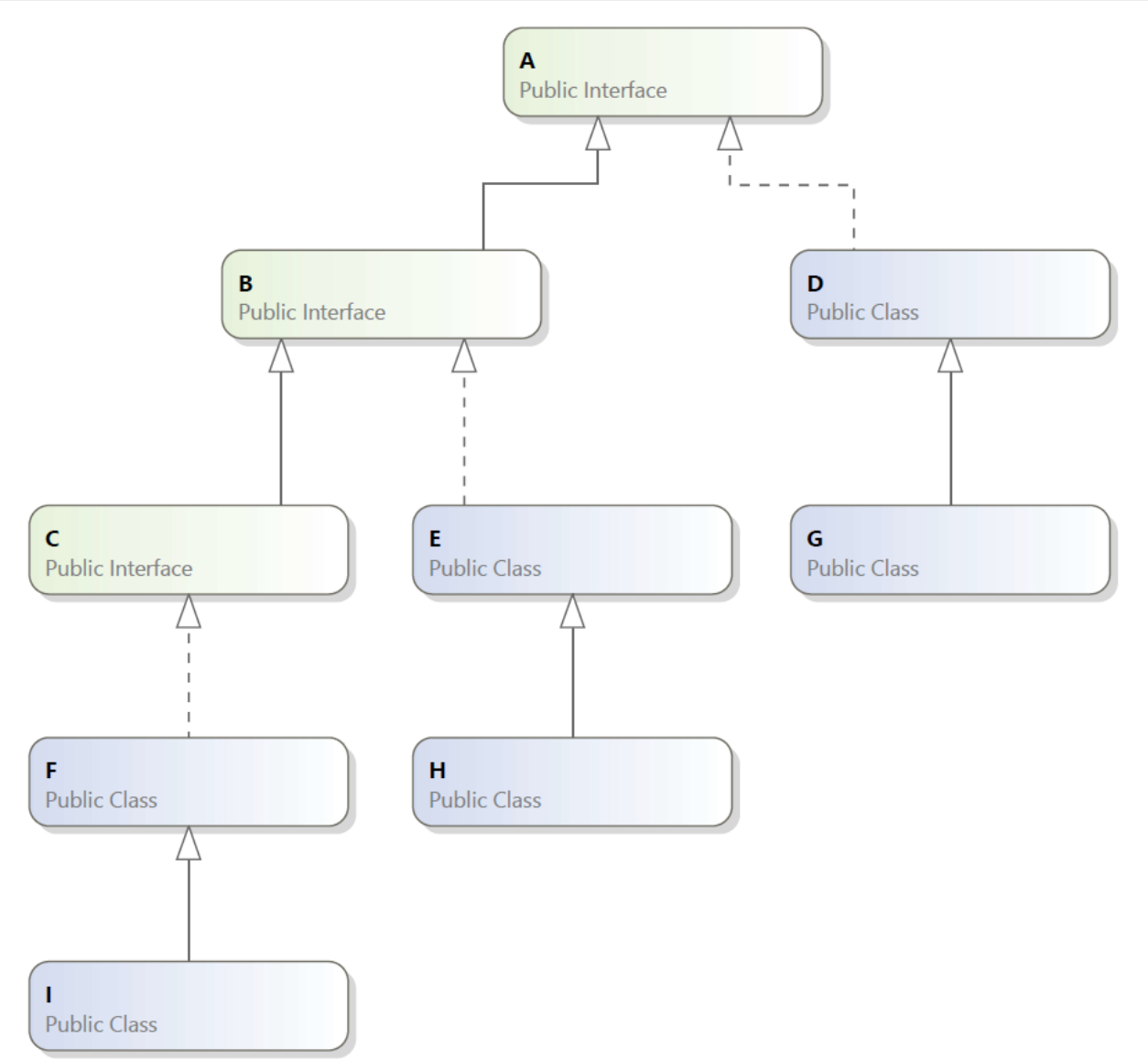
Prinsipper i Objektorientert Programmering

- Rett svar: Abstraction og Encapsulation
- Abstraction betyr å velge en måte å representere data på. Her har vi valgt å representere en Ball med radius og farge men utelatt andre egenskaper ved en virkelig ball, som materiale og lufttrykk.
- Ball er implementert slik at man kan ikke endre på radius og farge etter at et nytt Ball objekt er laget, dette gjøres ved å sette feltvariablene til private og er en type encapsulation.

Inheritance

- Rett svar: B,C,E,F,H,I

- Denne oppgaven spør hvilke typer variabler som kan tilordnes en variabel av typen B.
- Alle subtyper av B kan tilordnes variabel av type B.
- Tegn klassediagram og list opp alle klasser under B. (klassediagram på neste slide)


A
Public Interface
B
Public Interface
D
Public Class
C
Public Interface
E
Public Class
G
Public Class
F
Public Class
H
Public Class
I
Public Class

Collections.sort()

- Rett svar:
- Dersom T utvider en klasse som er sammenlignbar med seg selv, så kan List sorteres.
- "static" betyr at metoden kan kalles uten å først konstruere et Collections-objekt.
- Dersom T er sammenlignbar med seg selv, så kan List sorteres.

Teller

- På denne oppgaven skal du først lage et interface og så implementere det.
- Når du lager interface så må du tenke på gode metodenavn/kommentarer og rett returtype på metodene

Teller

```
public interface ICounter {

    /** @return the current count */
    public int getCount();

    /** Increases the count by one */
    public void increase();

    /** resets the counter to 0 */
    public void reset();
}
```

Teller

- Når du lager klasse som implementerer så må du velge rett feltvariabler.
- Her skal vi huske på et tall (så langt vi har kommet i tellingen) så derfor er en int naturlig.
- Tallet må settes til 0 når nytt objekt lages (enten konstruktør eller sette feltvariabelen direkte.
- For å sørge for at tallet ikke kan bli negativt setter vi feltvariabelen til private

```
public class Counter implements ICounter {

    private int count;
    public Counter() {
        count = 0;
    }
    @Override
    public int getCount() {
        return count;
    }
    @Override
    public void increase() {
        count++;
    }
    @Override
    public void reset() {
        count=0;
    }
}
```

Teller

- Det er en måte til som kan forårsake at teller blir negativ, det er integer overflow.
- I denne oppgaven trenger du ikke ta hensyn til det, men du kunne tatt hensyn til det på denne måten:

```
@Override
public void increase() {
    count++;
    if(count<0)
        throw new IllegalStateException(
        "Integer overflow caused counter to become negative");
}
```

ShoppingTest

- I denne oppgaven var dere gitt en Junit test som feilet og skulle finne ut hva som var feil.
- Det er en linje med assertEquals som feiler, denne metoden kaller på equals metoden til ShoppingItem
- Merk at ShoppingItem ikke har equals metode men bruker default fra Object.
- Dere skulle skrive en Equals metode
- Det enkleste er å la Eclipse generere en for dere. Source -> generate … equals()

ShoppingTest

```
@Override
public boolean equals(Object obj) {
    if (obj instanceof ShoppingItem){
        ShoppingItem other = (ShoppingItem) obj;
        if(!brand.equals(other.brand)) {
            return false;
        }
        if (!itemType.equals(other.itemType)) {
            return false;
        }
        return true;
    }
    return false;
}
```

Par

```
public interface IPair<A, B> {

    public A getFirst();

    public B getSecond();

}
```

Par

```
public interface IIntegerPair extends IPair<Integer, Integer> {

    public int sum();

}
```

Her kunne dere eventuelt laget sum som en default metode (begge deler er rett)

```
public default int sum() {
    return getFirst()+getSecond();
};
```

```java
public class IntegerPair implements IIntegerPair {

    private int first;
    private int second;

    public IntegerPair(int first, int second) {
        this.first = first;
        this.second = second;
    }

    @Override
    public Integer getFirst() {
        return first;
    }

    @Override
    public Integer getSecond() {
        return second;
    }

    @Override
    public int sum() {
        return getFirst() + getSecond();
    }
}
```

Fridge

- For å komme i gang med denne oppgaven må du lage en klasse som implementerer IFridge
- Så må du finne ut hvilke feltvariabler du trenger

```
public class Fridge implements IFridge {

    private List<FridgeItem> items;
    private int capacity;

    public Fridge(int capacity) {
        this.capacity = capacity;
        items = new ArrayList<FridgeItem>();
    }
```

Fridge

- Hvis du har rett feltvariabler så blir flere metoder enkle.

```
@Override
public int nItemsInFridge() {
    return items.size();
}

@Override
public int totalSize() {
    return capacity;
}

@Override
public void emptyFridge() {
    items.clear();
}
```

Fridge

- To metoder trengte en liten ekstra sjekk

```
@Override
public boolean placeIn(FridgeItem item) {
    if(items.size()>= totalSize())
        return false;
    return items.add(item);
}

@Override
public void takeOut(FridgeItem item) {
    if(!items.contains(item))
        throw new IllegalArgumentException("Item not in fridge");
    items.remove(item);
}
```

Fridge

- takeOut() metoden som tar inn en String er en standard for løkke for å finne de items med rett navn og så må man hente ut det item som først går ut på dato. Her kan det være lurt å huske på at FridgeItem er comparable.
- removeItems() er også ganske lett, men en fallgruve er at man kan ikke fjerne fra en liste mens man løkker igjennom samme listen så en hjelpemetode er smart.

```java
@Override
public FridgeItem takeOut(String itemName) {
    List<FridgeItem> rightItems = new ArrayList<FridgeItem>();
    for(FridgeItem item :items) {
        if(item.getName().equals(itemName)) {
            rightItems.add(item);
        }
    }
    if(rightItems.isEmpty()) {
        throw new IllegalArgumentException("Item not in fridge");
    }
    FridgeItem found = Collections.min(rightItems);
    takeOut(found);
    return found;
}
```

```
@Override
public List<FridgeItem> removeExpiredFood() {
    List<FridgeItem> expItems = getExpiredItems();
    items.removeAll(expItems);
    return expItems;
}

private List<FridgeItem> getExpiredItems() {
    List<FridgeItem> expItems = new ArrayList<FridgeItem>();
    for(FridgeItem item : items) {
        if(item.hasExpired()){
            expItems.add(item);
        }
    }
    return expItems;
}
```

Vaksineplan oppg. 1

Denne oppgaven ble litt for lett for dere….

```
public class Patient implements Comparable<Patient> {

    @Override
    public int compareTo(Patient o) {
        if (this.underlyingConditionGrade != o.underlyingConditionGrade) {
            return Integer.compare(o.underlyingConditionGrade.getValue(),
                    this.underlyingConditionGrade.getValue());
        } else {
            return Integer.compare(o.age, this.age);
        }
    }
```

Vaksineplan oppg. 2

```
public class Moderna extends Vaccine {

    public Moderna(LocalDate date) {
        super(date);
    }
    @Override
    public String getName() {
        return "Moderna";
    }
}
```

Vaksineplan oppg. 2

```
public class Pfizer extends Vaccine {

    public Pfizer(LocalDate deliveryDate) {
        super(deliveryDate);
    }
    @Override
    public String getName() {
        return "Pfizer";
    }
}
```

Vaksineplan oppg. 3

- Her skulle du ordne to lister (pasienter og vaksiner) og skrive ut par.
- Det enkleste er kanskje å sortere listene

```
public static void main(String[] args) {
    List<Patient> patients = getPatients();
    List<Vaccine> vaccines = getVaccines();

    Collections.sort(patients);
    Collections.sort(vaccines);

    for (int i = 0; i < vaccines.size(); i++) {
        assignVaccine(patients.get(i), vaccines.get(i));
    }
}
```

Vaksineplan oppg. 3

- Alternativ løsning (med generisk hjelpemetode)

```
public static void main(String[] args) {
    List<Patient> patients = getPatients();
    List<Vaccine> vaccines = getVaccines();

    while(!vaccines.isEmpty()) {
        Patient patient = removeMin(patients);
        Vaccine vaccine = removeMin(vaccines);
        assignVaccine(patient, vaccine);
    }
}

public static <T extends Comparable<? super T>> T removeMin(List<T> elements) {
    T min = Collections.min(elements);
    elements.remove(min);
    return min;
}
```