

<https://tinyurl.com/inf101ccv24>



INF101 kræsjskurs

Objektorientert programmering

av Alexander Höpner og Gisle Kvamme

Plan for dagen

- Java som språk
 - Typer, lister etc.
 - Klasser og objekter
 - Arv
 - Dokumentasjon
 - Testing
- Prinsipper i objektorientert programmering
- Grafikk
- Eksamensoppgaver
- Praktisk del

Java som språk

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

- Objektorientert språk
- Må kompilere før det kjøres
- Statiske typer

Eksempler i python

```
x = 21
x += 1 # x = 22
```

```
s = "Hello"
f = 3.14
b = True
```

```
p = True
q = False
```

```
a = p and q
b = p or q
c = not p
```

```
l = [1, 2, 3, 4, 5]
l.append(6)
l.remove(3)
l[0] = 10
```

```
if x > 0:
    print("x is positive")
elif x < 0:
    print("x is negative")
else:
    print("x is zero")
```

```
for i in range(10):
    print(i)
```

```
for i in range(10, 20):
    print(i)
```

```
for i in range(10, 20, 2):
    print(i)
```

```
for i in l:
    print(i)
```

```
i = 0
while i < 10:
    print(i)
    i += 1
```

```
def add(x, y):
    return x + y
```

Eksempler i java

```
int x = 21;  
x += 1; // x = 22
```

```
String s = "Hello";  
double f = 3.14;  
boolean bn = true;
```

```
boolean p = true;  
boolean q = false;
```

```
boolean a = p && q;  
boolean b = p || q;  
boolean c = !p;
```

```
ArrayList<Integer> l = new ArrayList<>();  
l.add(1);  
l.remove(0);  
l.get(0);
```

```
if (x > 0) {  
    System.out.println("x is positive");  
} else if (x < 0) {  
    System.out.println("x is negative");  
} else {  
    System.out.println("x is zero");  
}
```

```
for (int i = 0; i < 10; i++) {  
    System.out.println(i);  
}
```

```
int i = 0;  
while (i < 10) {  
    System.out.println(i);  
    i++;  
}
```

```
int add(int x, int y) {  
    return x + y;  
}
```

Primitive typer og Refererte typer

- boolean
- byte
- short
- int
- long
- float
- double
- char
- Boolean
- Byte
- Short
- Integer
- Long
- Float
- Double
- Character

Minnet

		minneadresse	verdi	
stack	{	int x	63999999936	12345
		Integer y	639999999872	196
			639999999808	
		...		
heap	{		...	
			256	
			196	12345
			128	
			64	
			0	

Viewer does not support full SVG 1.1

List/Arrays

```
int[] a = {1, 2, 3};
```

```
ArrayList<Integer> b = new ArrayList<Integer>();  
b.add(1);  
b.add(2);  
b.add(3);
```

```
List<Integer> c = new ArrayList<Integer>();  
c.add(1);  
c.add(2);  
c.add(3);
```


Hva vil det si at noe er statisk?

- Begrepet
- Noe som tilhører klassen og ikke objektet
- Helt greit med statiske metoder
- **ALDRI HA STATISKE VARIABLER (nesten)**
 - Statiske konstanter
 - Defaults

Klasser og objekter

```
public class Sheep {
```

```
    private static final String BREED = "Sheep";  
    private String name;
```

```
    public Sheep(String name) {  
        this.name = name;  
    }
```

```
    public Sheep() {  
        this(BREED);  
    }
```

```
    public String getBreed() {  
        return breed;  
    }
```

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

```
    public String makeSound() {  
        return "Baaaah!";  
    }
```

Klassekonstant

Instansvariabel

Konstruktør

Instansmetoder

Eksamensoppgave V22

```
public class AIPlayer {  
    private String name = "AIPlayer";  
  
    public void talk() {  
        System.out.println("I am a robot brrrr");  
    }  
  
    @Override  
    public String toString() {  
        return name;  
    }  
}
```

Hvor mange argumenter forventer konstruktøren?

Grensesnitt

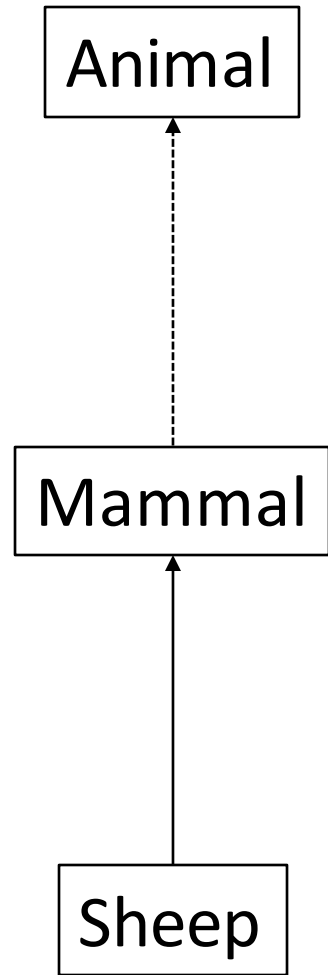
```
public interface Animal {  
    String makeSound();  
    String getBreed();  
    String getName();  
}
```

Abstrakte metode-deklarasjoner

Arv

```
public abstract class Mammal {  
    public boolean laysEggs() {  
        return false;  
    }  
}
```

Arv



Eksamensoppgave V22

```
public class Person {  
  
    private String name;  
    private int yearOfBirth;  
  
    public Person(String name, int yearOfBirth) {  
        this.name = name;  
        this.yearOfBirth = yearOfBirth;  
    }  
  
    public String getName() {  
        return this.name;  
    }  
  
    public int getYearOfBirth() {  
        return this.yearOfBirth;  
    }  
}  
  
public static void main(String[] args) {  
    Person x = new Person("Tor", 1999);  
    Person y = new Person("To" + "r", 1999);  
    System.out.println(x == y);  
}
```

Eksamensoppgave V22

```
public class Person {  
  
    private String name;  
    private int yearOfBirth;  
  
    public Person(String name, int yearOfBirth) {  
        this.name = name;  
        this.yearOfBirth = yearOfBirth;  
    }  
  
    public String getName() {  
        return this.name;  
    }  
  
    public int getYearOfBirth() {  
        return this.yearOfBirth;  
    }  
  
    @Override  
    public boolean equals(Object other) {  
        if (!(other instanceof Person))  
            return false;  
  
        Person otherPerson = (Person) other;  
        return this.name.equals(otherPerson.name) && this.yearOfBirth == otherPerson.yearOfBirth;  
    }  
}  
  
public static void main(String[] args) {  
    Person x = new Person("Tor", 1999);  
    Person y = new Person("To" + "r", 1999);  
    System.out.println(x.equals(y));  
}
```


Java.lang.Object

- Alle objekter arver fra Object
- Standardmetoder:
 - hashCode()
 - equals(Object obj)
 - toString()

Likhet

- equals()
- Standardimplementasjon fra Object (java.lang)
- Må som regel lage egen
 - Sammenlign variabler
- Krav til likhet:
 - Refleksivitet
 - Symmetri
 - Transitivitet
 - Konsistent
 - Null-sikkerhet
 - hash-sikkerhet

Sammenlikning

- Comparable<T>
 - compareTo(T o)
- Total orden
 - Refleksiv
 - Transitiv
 - Antisymmetrisk
 - Total

Arv vs komposisjon

- Klasser som arver «er en»
- Klasser som benytter komposisjon «har en»

1. Implementer en kortbunke-klasse *InheritedArrayCardPile* ved bruk av **arv** fra *ArrayList*.
2. Implementer en kortbunke-klasse *ComposedArrayCardPile* ved bruk av **komposisjon** av *ArrayList*.

For begge klassene skal du:

- Implementere en metode *createFullDeck()* som returnerer en ny bunke med kort. Metoden skal returnere et nytt kortbunke-objekt med alle de 52 forskjellige kortene. Burde denne metoden være static eller non-static?
- Implementere en metode med signatur *boolean add(Card)* som legger til et kort i bunken. Metoden skal kaste *IllegalArgumentException* dersom noen forsøker å legge til null, men skal ellers legge til kortet i bunken og alltid returnere true (rart, men slik er spesifikasjonene). Burde denne metoden være static eller non-static?

Generiske typer

```
public class Box<T> {  
    List<T> contents = new ArrayList<T>();  
  
    public void add(T thing) {  
        contents.add(thing);  
    }  
  
    public void remove(T thing) {  
        contents.remove(thing);  
    }  
}
```

```
public static void main(String[] args) {  
    Toy toyCar = new Toy("Car");  
    Toy toyDoll = new Toy("Doll");  
    Toy toyTrain = new Toy("Train");  
  
    Box<Toy> toyBox = new Box<Toy>();  
    toyBox.add(toyCar);  
    toyBox.add(toyDoll);  
    toyBox.add(toyTrain);  
}
```

JavaDoc

```
/**  
 * A box can contain anything  
 */  
public Box() {  
    contents = new ArrayList<T>();  
}
```

```
/**  
 * Put something in the box  
 * @param thing to put in the box  
 */  
public void add(T thing) {  
    contents.add(thing);  
}
```

```
void com.TEKNA.course.classes.species.generics.Box.add(T thing)
```

Put something in the box

- **Parameters:**
 - **thing** to put in the box

Testing

```
public class BoxTest {  
  
    Box<Toy> toyBox;  
  
    @BeforeEach  
    public void setUp() {  
        toyBox = new Box<Toy>();  
    }  
  
    @Test  
    public void testAdd() {  
        Toy toyCar = new Toy("Car");  
        Toy toyDoll = new Toy("Doll");  
        Toy toyTrain = new Toy("Train");  
        toyBox.add(toyCar);  
        toyBox.add(toyDoll);  
        toyBox.add(toyTrain);  
        assertEquals(3, toyBox.contents.size());  
    }  
}
```

Objektorientert programming

Prinsipper

Abstraksjon

- Vanskelig å lage en 100% tro kopi
- Fokus på de viktigste egenskapene
- Hva trengs i ditt prosjekt?

Eksempel på abstraksjon

```
public class Sheep {  
  
    private String name;  
  
    public Sheep(String name) {  
        this.name = name;  
    }  
  
    public String getBreed() {  
        return breed;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public String makeSound() {  
        return "Baaaah!";  
    }  
}
```

Innkapsling

- Et prosjekt er delt opp i pakker
- Tilgangsmodifikatorer
- Restriktive typer
- Prøv å være så streng som mulig
- Get-ere og set-ere

Modifikator	Klasse	Pakke	Subklasse	Alle
<code>public</code>	Ja	Ja	Ja	Ja
<code>protected</code>	Ja	Ja	Ja	Nei
~ ingen modifikator ~ (aka «pakke-privat»/standard/default)	Ja	Ja	Nei	Nei
<code>private</code>	Ja	Nei	Nei	Nei

Eksempel på innkapsling

```
public class Text implements ReadOnlyText {  
  
    private String text;  
  
    public Text(String text) {  
        this.text = text;  
    }  
  
    public void write(String text) {  
        this.text = text;  
    }  
  
    @Override  
    public String read() {  
        return text;  
    }  
}  
  
public interface ReadOnlyText {  
  
    String read();  
}  
  
public static void main(String[] args) {  
    ReadOnlyText text = new Text("Hello World!");  
    text.read();  
  
    // text.write("Hello World!"); // Not possible  
}
```

Polymorfisme

- Når flere klasser kan brukes på en uniform måte
- Vanlig ved bruk av f. eks:
 - Grensesnitt
 - Arv
- Statisk polymorfisme:
 - Flere metoder med samme navn
 - Ulik parameterliste
- Dynamisk polymorfisme
 - @Override

Eksempel på statisk polymorfisme

```
public class Math {  
  
    public static int add(int a, int b) {  
        return a + b;  
    }  
  
    public static double add(double a, double b) {  
        return a + b;  
    }  
  
    public static int add(int a, int b, int c) {  
        return a + b + c;  
    }  
}
```

Eksempel på dynamisk polymorfisme

```
public static void main(String[] args) {  
  
    List<Animal> animals = new ArrayList<>();  
    animals.add(new Sheep("Dolly"));  
    animals.add(new Dog("Fido"));  
    animals.add(new Cat("Garfield"));  
    animals.add(new Platypus("Perry"));  
  
    for (Animal animal : animals) {  
        animal.makeSound();  
    }  
}
```

Mutabilitet

- Sier noe om hvorvidt verdier kan endres
- Noe er muterbart dersom det kan endres
 - Set-metoder
- Noe er uforandelig dersom det ikke kan endres
 - Final
 - Variabler – konstanter
 - Metoder – Kan ikke overskrives av subklasser
 - Klasser – Kan ikke utvides

Eksempel på mutabilitet

```
public class Coordinate {  
  
    private int x;  
    private int y;  
  
    public Coordinate(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public void update(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Eksempel på immutabilitet

```
public record Coordinate(int x, int y) {}
```

Hva er records?

```
public final class Coordinate {
```

```
    private final int x;  
    private final int y;
```

```
    public Coordinate(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }
```

```
    public Coordinate(Coordinate coordinate) {  
        this(coordinate.x, coordinate.y);  
    }
```

```
    public int x() {  
        return x;  
    }
```

```
    public int y() {  
        return y;  
    }
```

```
}
```

```
@Override
```

```
public boolean equals(Object obj) {  
    if (!(obj instanceof Coordinate)) {  
        return false;  
    }  
    Coordinate other = (Coordinate) obj;  
    return x == other.x && y == other.y;  
}
```

```
@Override
```

```
public int hashCode() {  
    return Objects.hash(x, y);  
}
```

```
@Override
```

```
public String toString() {  
    return "Coordinate [x=" + x + ", y=" + y + "]";  
}
```

Vær obs på mutabilitet!

```
public class TextUtil {
    private final StringBuilder text;

    public TextUtil() {
        text = new StringBuilder("Hello World!");
    }

    public StringBuilder getText() {
        return text;
    }
}

public class OutsideSource {
    public static void main(String[] args) {
        TextUtil util = new TextUtil();

        StringBuilder text = util.getText();

        text.append(" Goodbye World!"); //!
    }
}
```

```

1 ▼ public class Person {
2
3     private String name;
4     private int yearOfBirth;
5
6 ▼ public Person(String name, int yearOfBirth) {
7     this.name = name;
8     this.yearOfBirth = yearOfBirth;
9 ▲ }
10
11 ▼ public String getName() {
12     return this.name;
13 ▲ }
14
15 ▼ public int getYearOfBirth() {
16     return this.yearOfBirth;
17 ▲ }
18
19 ▲ }

```

Hvilke av følgende konsepter brukes i klassen *Person*?

Instansmetoder

Generiske typer

Uforanderlighet (immutability)

Komposisjon

Feltvariabler

Abstraksjon

Polymorfisme

Innkapsling

Eksempeloppgave v22

Iterator og iterable

- Gir oss en måte å se på alle objekter i en samling
- Iterator
 - Et objekt som blar gjennom en samling av objekter
- Iterable
 - Et objekt man kan bla gjennom flere ganger
 - Lager flere iterator-objekter

Iterator

```
public interface Iterator<E> {  
    /**  
     * Returns {@code true} if the iteration has more elements.  
     * (In other words, returns {@code true} if {@link #next} would  
     * return an element rather than throwing an exception.)  
     *  
     * @return {@code true} if the iteration has more elements  
     */  
    boolean hasNext();  
  
    /**  
     * Returns the next element in the iteration.  
     *  
     * @return the next element in the iteration  
     * @throws NoSuchElementException if the iteration has no more elements  
     */  
    E next();  
}
```

Iterable

```
public interface Iterable<T> {  
    /**  
     * Returns an iterator over elements of type {@code T}.  
     *  
     * @return an Iterator.  
     */  
    Iterator<T> iterator();  
}
```


Iterable hierarkiet

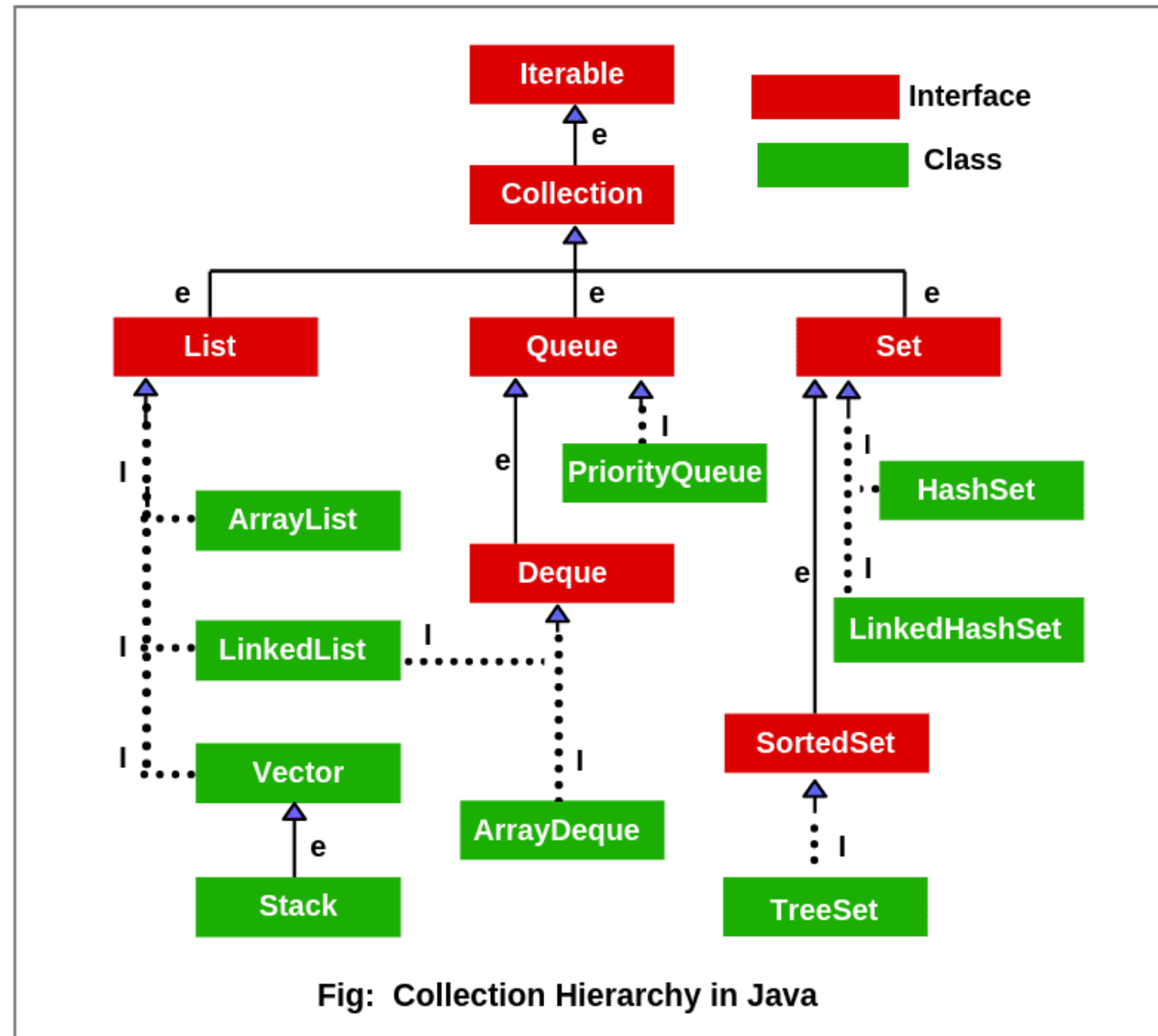


Fig: Collection Hierarchy in Java

Mer om dette i INF102

Hentet fra [Scientecheasy](https://www.scientecheasy.com/)

Eksamensoppgave V22

```
public class Book {  
    private List<Page> pages;  
  
    public Book(List<Page> pages) {  
        this.pages = pages;  
    }  
  
    public int length() {  
        return pages.size();  
    }  
  
    public Page getPage(int pageNumber) {  
        if (pageNumber >= length())  
            throw new IndexOutOfBoundsException("Index out of bounds");  
  
        return pages.get(pageNumber);  
    }  
}  
  
public static void main(String[] args) {  
    Book haroldPoter = new Book(Arrays.asList(  
        new Page("Why doesn't Voldemort wear glasses?"),  
        new Page("Nobody nose.")));  
  
    for (Page page : haroldPoter) {  
        String text = page.getText();  
        System.out.println(text);  
    }  
}
```

Eksamensoppgave V22

```
public class Book implements Iterable<Page> {  
    private List<Page> pages;  
  
    public Book(List<Page> pages) {  
        this.pages = pages;  
    }  
  
    public int length() {  
        return pages.size();  
    }  
  
    public Page getPage(int pageNumber) {  
        if (pageNumber >= length())  
            throw new IndexOutOfBoundsException("Index out of bounds");  
  
        return pages.get(pageNumber);  
    }  
  
    @Override  
    public Iterator<Page> iterator() {  
        return pages.iterator();  
    }  
}  
  
public static void main(String[] args) {  
    Book haroldPoter = new Book(Arrays.asList(  
        new Page("Why doesn't Voldemort wear glasses?"),  
        new Page("Nobody nose.")));  
  
    for (Page page : haroldPoter) {  
        String text = page.getText();  
        System.out.println(text);  
    }  
}
```


Designmønstre I

Type	Funksjon:
Creational	Kontroll over opprettelsen av objekter.
Structural	Kontroll over komposisjonen av klasser og objekter.
Behavioral	Kontroll over samspillet mellom klasser og objekter.

Designmønstre II

Type	Eksempler
Creational	Factory, singleton, builder
Structural (NB!)	Adapter, decorator, proxy
Behavioral	Iterator, observer, memento, eventbus

Factory

```
public abstract class ShapeFactory {  
    public static final Shape createShape(String shapeType) {  
        if (shapeType.equals("Circle")) {  
            return new Circle();  
        } else if (shapeType.equals("Rectangle")) {  
            return new Rectangle();  
        } else if (shapeType.equals("Triangle")) {  
            return new Triangle();  
        } else {  
            throw new IllegalArgumentException("Invalid shape type");  
        }  
    }  
}
```

Navn Navnesen
Navnegata 5
9000 Tromsø

014
199
A7B

EventBus

```
public class EventBus {  
    private List<EventHandler> eventHandlers = new ArrayList<>();  
  
    void register(EventHandler handler) {  
        eventHandlers.add(handler);  
    }  
  
    void post(Event event) {  
        for (EventHandler eventHandler : eventHandlers) {  
            eventHandler.handle(event);  
        }  
    }  
}  
  
public interface EventHandler {  
    void handle(Event e);  
}
```

```
public class Sender {  
  
    private EventBus bus;  
  
    public Sender(EventBus bus) {  
        this.bus = bus;  
    }  
  
    void send(int one, int two) {  
        System.out.println("Sending message: " + one + " " + two);  
        bus.post(new Event(one, two));  
    }  
  
}  
  
public record Event(int one, int two) {}
```

```
public class Adds implements EventHandler {
    public Adds(EventBus bus) {
        bus.register(this);
    }

    @Override
    public void handle(Event e) {
        System.out.println(e.one()+e.two());
    }
}
```

```
public class Subtracts implements EventHandler {
    ...

    @Override
    public void handle(Event e) {
        System.out.println(e.one()-e.two());
    }
}
```

```
public class Multiplies implements EventHandler {
    ...

    @Override
    public void handle(Event e) {
        System.out.println(e.one()*e.two());
    }
}
```

```
public static void main(String[] args) {  
    EventBus eventBus = new EventBus();  
    Sender a = new Sender(eventBus);  
  
    EventHandler one = new Adds(eventBus);  
    EventHandler two = new Subtracts(eventBus);  
    EventHandler three = new Multiplies(eventBus);  
  
    a.send(5, 3);  
}
```

Sending message: 5 3

8

2

15

Observer og Observable

```
40 public interface Observer {
41     /**
42      * This method is called whenever the observed object is changed. An
43      * application calls an {@code Observable} object's
44      * {@code notifyObservers} method to have all the object's
45      * observers notified of the change.
46      *
47      * @param o    the observable object.
48      * @param arg  an argument passed to the {@code notifyObservers}
49      *             method.
50      */
51     void update(Observable o, Object arg);
52 }
53
```

Observer og Observable

```
public class Observer<E> {  
    private E observedValue;  
  
    public void update(E value) {  
        observedValue = value;  
    }  
  
    public Object getObservedValue() {  
        return observedValue;  
    }  
}
```

```
public class Observable<E> {  
    List<Observer<E>> observers = new ArrayList<>();  
  
    public void update(E value) {  
        for (Observer<E> obs : observers) {  
            obs.update(value);  
        }  
        System.out.println("Updated the valu...");  
    }  
  
    public void addObserver(Observer<E> observer) {  
        observers.add(observer);  
    }  
}
```

Run | Debug

```
public static void main(String[] args) {  
    Observable<Integer> observableNumber = new Observable<>();  
    Observer<Integer> obs1 = new Observer<>();  
    Observer<Integer> obs2 = new Observer<>();  
    Observer<Integer> obs3 = new Observer<>();  
  
    observableNumber.addObserver(obs1);  
    observableNumber.addObserver(obs2);  
    observableNumber.addObserver(obs3);  
    System.out.println(x:"////////////////////");  
  
    observableNumber.update(value:42);  
  
    System.out.println(obs1.getObservedValue());  
    System.out.println(obs2.getObservedValue());  
    System.out.println(obs3.getObservedValue());  
    System.out.println(x:"////////////////////");  
    observableNumber.update(value:50);  
    System.out.println(x:"////////////////////");  
    System.out.println(obs1.getObservedValue());  
    System.out.println(obs2.getObservedValue());  
    System.out.println(obs3.getObservedValue());  
}
```

>

```
////////////////////  
Updated the value to 42  
42  
42  
42  
////////////////////  
Updated the value to 50  
////////////////////  
50  
50  
50
```

-

Creational:

- Factory
- Builder
- Prototype
- Singleton

-

Structural:

- Adapter
- Loose coupling
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behaviorial:

- Chain of responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template method
- Visitor

Eksamen V21

```
public class Grid<T extends ShoppingItem> {  
  
    private List<T> cells;  
    private int columns;  
    private int rows;  
  
    public Grid(int rows, int columns, T initElement) {  
        if (rows <= 0 || columns <= 0)  
            throw new IllegalArgumentException();  
        this.columns = columns;  
        this.rows = rows;  
        cells = new ArrayList<>(columns * rows);  
        for (int i = 0; i < columns * rows; ++i) {  
            cells.add(initElement);  
        }  
    }  
}
```

Hvilke datatyper kan denne Grid-klassen bruke (hvilke datatyper kan typeparameteren T være)?

ShoppingItem og subtyper av ShoppingItem

```
public class Ball {  
  
    private double radius;  
    private Color color;  
  
    public Ball(double radius, Color color) {  
        this.radius = radius;  
        this.color = color;  
    }  
  
    public Color getColor() {  
        return color;  
    }  
  
    public double radius() {  
        return radius;  
    }  
  
}
```

Hvilke av de følgende konseptene brukes i klassen *Ball*?

Velg ett eller flere alternativer

- ☐ Polymorphism
- ☐ Overriding
- ☐ Abstraction
- ☐ Encapsulation
- ☐ Generics
- ☐ Inheritance

Encapsulation og abstraction

I denne oppgaven tar vi for oss en kodebase som består av de følgende interfacene og klassene:

- et interface A
- et interface B som utvider A
- et interface C som utvider B
- en klasse D som implementerer A
- en klasse E som implementerer B
- en klasse F som implementerer C
- en klasse G som utvider D
- en klasse H som utvider E
- en klasse I som utivder F

I et program oppretter vi en variabel av typen B:

```
B minVariabel;
```

Hvilke typer kan et objekt ha for at det skal kunne legges i variabelen?

Med andre ord, hvilke typer kan en annen variabel `a` ha slik at følgende er lov:

```
minVariabel = a;
```

Redusér problemet før du løser det

I et program oppretter vi en variabel av typen B:

```
B minVariabel;
```

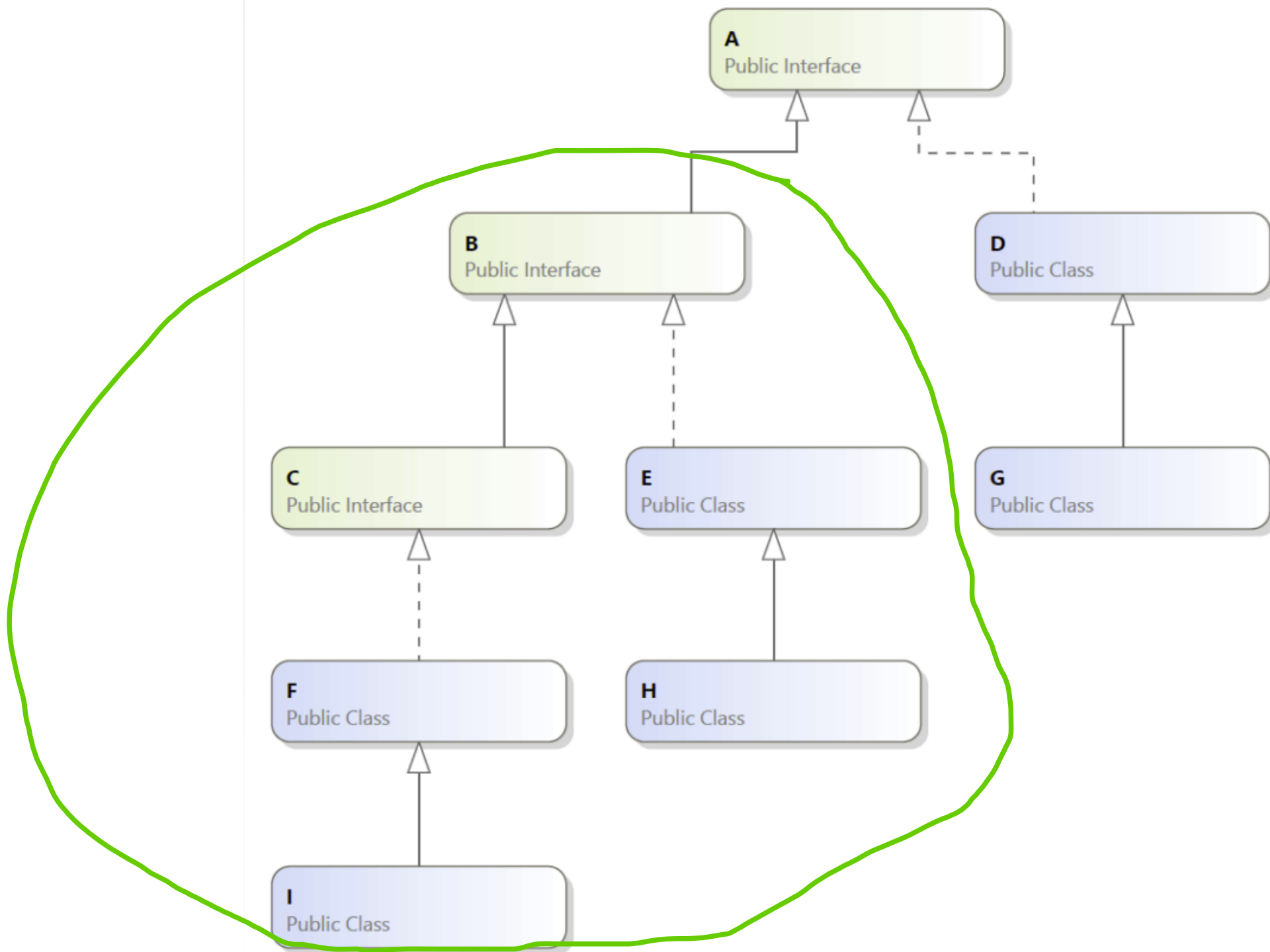
Hvilke typer kan et objekt ha for at det skal kunne legges i variabelen?

Med andre ord, hvilke typer kan en annen variabel a ha slik at følgende er lov:

```
minVariabel = a;
```



MinVariabel kan være av typen B eller subtyper av B. Dvs. Alt som arver eller implementer fra B.



- et interface A ✗
- et interface B som utvider A ✓
- et interface C som utvider B ✓
- en klasse D som implementerer A ✗
- en klasse E som implementerer B ✓
- en klasse F som implementerer C ✓
- en klasse G som utvider D ✗
- en klasse H som utvider E ✓
- en klasse I som utivder F ✓

Typesignaturen til Collections.sort

For sortering av en liste i java kan en bruke metoden *Collections.sort* som har den følgende signaturen:

```
static <T extends Comparable<? super T>> void sort(List<T> list)
```

Velg de påstandene som er riktig for metoden *sort*.

Velg ett eller flere alternativer

- ☒ "static" betyr at metoden kan kalles uten å først konstruere et Collections-objekt.
- ☐ Dersom en klasse som utvider T er sammenlignbar med seg selv, så kan List sorteres.
- ☒ Dersom T er sammenlignbar med seg selv, så kan List sorteres.
- ☐ "static" betyr at klassen List må være statisk for at en liste skal kunne sorteres.
- ☐ Klassen List må utvide klassen Comparable for at listen skal kunne sorteres.
- ☒ Dersom T utvider en klasse som er sammenlignbar med seg selv, så kan List sorteres.

I denne oppgaven skal du, for hver av variabeldeklarasjonene under velge metodekallene som er lovlige i Java (dvs. de som ikke gir kompileringsfeil), gitt klassene og grensesnittene i PDFen. Du kan regne med at alle nødvendige metoder er implementert i klassene. 0,5 for riktig svar, 5 poeng for alt rett.

```
interface IDuck {
    public void quack();
}

interface IFrog extends IDuck {
    public void jump();
}

class Mallard implements IDuck {
    public void swim() {
    }

    // ...
}

class Toad {
    public void walk() { /* ... */ };
}

class Frog extends Toad implements IFrog {
    // ...
}
```

```
interface IDuck {
    public void quack();
}

interface IFrog extends IDuck {
    public void jump();
}

class Mallard implements IDuck {
    public void swim() {
    }

    // ...
}

class Toad {
    public void walk() { /* ... */ };
}

class Frog extends Toad implements IFrog {
    // ...
}
```

IDuck a = new Mallard();
Velg et eller flere alternativer

- ☐ a.jump()
- ☒ a.quack()
- ☐ a.swim()
- ☐ a.walk()

```
interface IDuck {  
    public void quack();  
}  
  
interface IFrog extends IDuck {  
    public void jump();  
}  
  
class Mallard implements IDuck {  
    public void swim() {  
    }  
  
    // ...  
}  
  
class Toad {  
    public void walk() { /* ... */ };  
}  
  
class Frog extends Toad implements IFrog {  
    // ...  
}
```

IFrog b = new Frog();

✓ ☐ b.jump()

✓ ☐ b.quack()

☐ b.swim()

☐ b.walk()

```
interface IDuck {
    public void quack();
}

interface IFrog extends IDuck {
    public void jump();
}

class Mallard implements IDuck {
    public void swim() {
    }

    // ...
}

class Toad {
    public void walk() { /* ... */ };
}

class Frog extends Toad implements IFrog {
    // ...
}
```

Mallard c = new Mallard();
Velg ett eller flere alternativer

- ☐ c.jump()
- ☒ c.quack()
- ☒ c.swim()
- ☐ c.walk()

```
interface IDuck {  
    public void quack();  
}  
  
interface IFrog extends IDuck {  
    public void jump();  
}  
  
class Mallard implements IDuck {  
    public void swim() {  
    }  
  
    // ...  
}  
  
class Toad {  
    public void walk() { /* ... */ };  
}  
  
class Frog extends Toad implements IFrog {  
    // ...  
}
```

IDuck d = new Frog();
Velg ett eller flere alternativer

- ☐ d.jump()
- ☒ d.quack()
- ☐ d.swim()
- ☐ d.walk()

```
interface IDuck {  
    public void quack();  
}  
  
interface IFrog extends IDuck {  
    public void jump();  
}  
  
class Mallard implements IDuck {  
    public void swim() {  
    }  
  
    // ...  
}  
  
class Toad {  
    public void walk() { /* ... */ };  
}  
  
class Frog extends Toad implements IFrog {  
    // ...  
}
```

Frog e = new Frog();
Velg ett eller flere alternativer

- ☒ e.jump()
- ☒ e.quack()
- ☐ e.swim()
- ☒ e.walk()

Grafikk

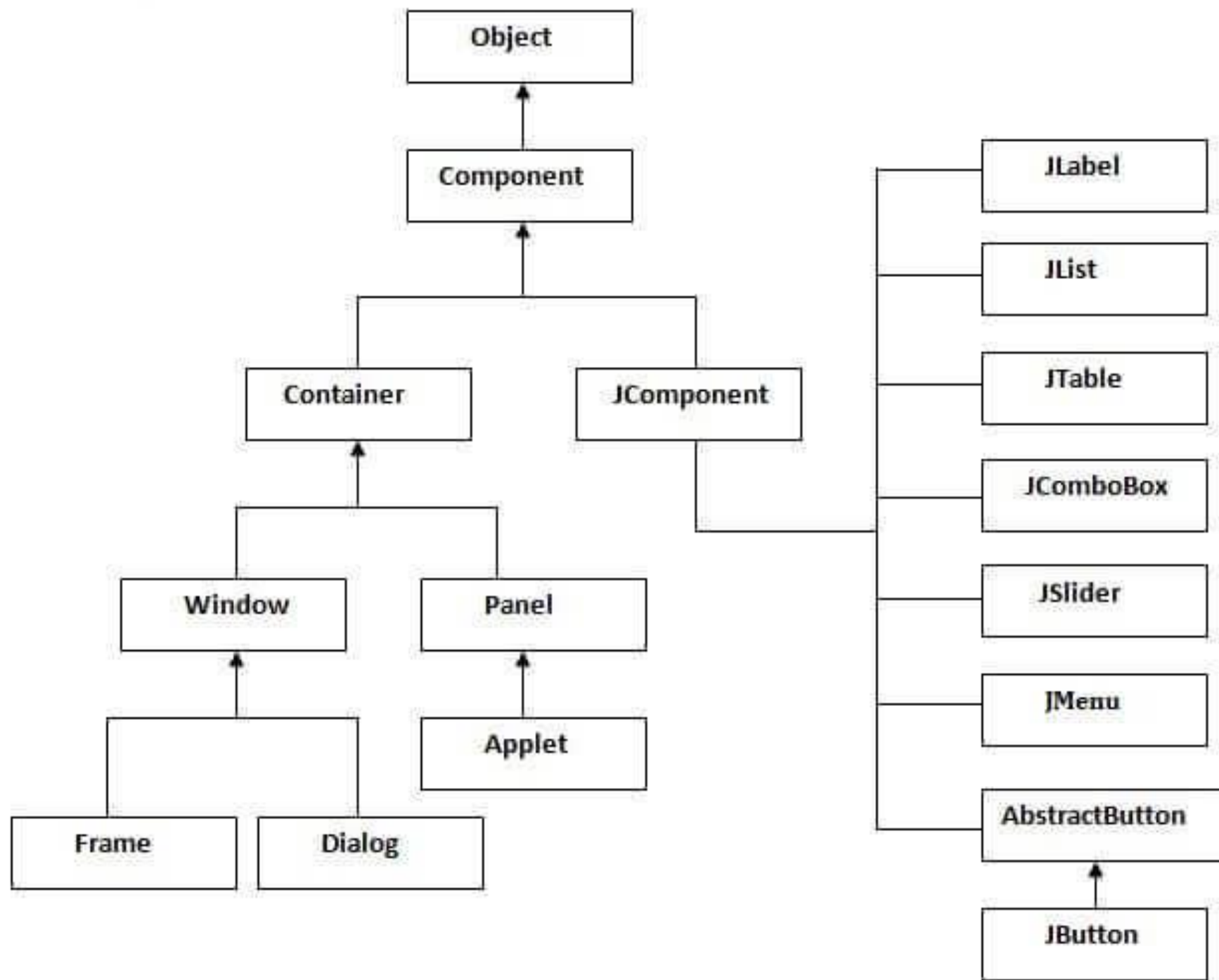
En repitisjon av Swing, samt en implementasjon av en enkel GUI-applikasjon

Swing

- Utvidelse av java.awt, Javas originale bibliotek for GUI, inneholder bl.a. Graphics, støtte for farger, geometriske former.
- Lettere å anvende enn awt
- Platformsuavhengige applikasjoner
- God støtte for MVC-inndeling

Model-View-Controller

- Et designmønster for GUI-applikasjoner
- Gir en modulær og innkapslet inndeling av programmet
- En modell håndterer "kjernelogikken"
- En kontrollør interagerer med denne modellen
- En visning representerer modellen
- Denne separasjonen gjør det lettere å utvikle i, feilsøke, og utbygge videre på de ulike komponentene



JPanel

- En "container" som kan holde på diverse grafiske komponenter, særlig Graphics-objekter. (som kommer fra java.awt).
- I vår applikasjon skal vi lage en tegneklasse som vil arve fra denne klassen. Dette gjør at vi enkelt kan synliggjøre modellen.

JFrame

- Tegneklassen vår, som er et JPanel, vil kunne legges til en "ramme" av typen JFrame.
- JFrame er en komponent på "toppnivå", og vil være selve programvinduet til applikasjonen.
- Dermed vil vår JFrame "holde" på en instans av vår tegneklasse, og vil være i stand til å representere modellen.

MouseAdapter

- Ikke en del av Swing, men java.awt
- Swing utvider dog awt, og dette tillater at Swing-komponenter kan benytte seg av f.eks. awts MouseAdapter og KeyListener
- Denne vil gjøre slik at vinduet vil være "klikkbart"

Point2D

- Ikke en del av Swing, men java.awt
- En abstrakt klasse som brukes for å lagre 2D-koordinater. Som regel brukes Point som den konkrete klassen.

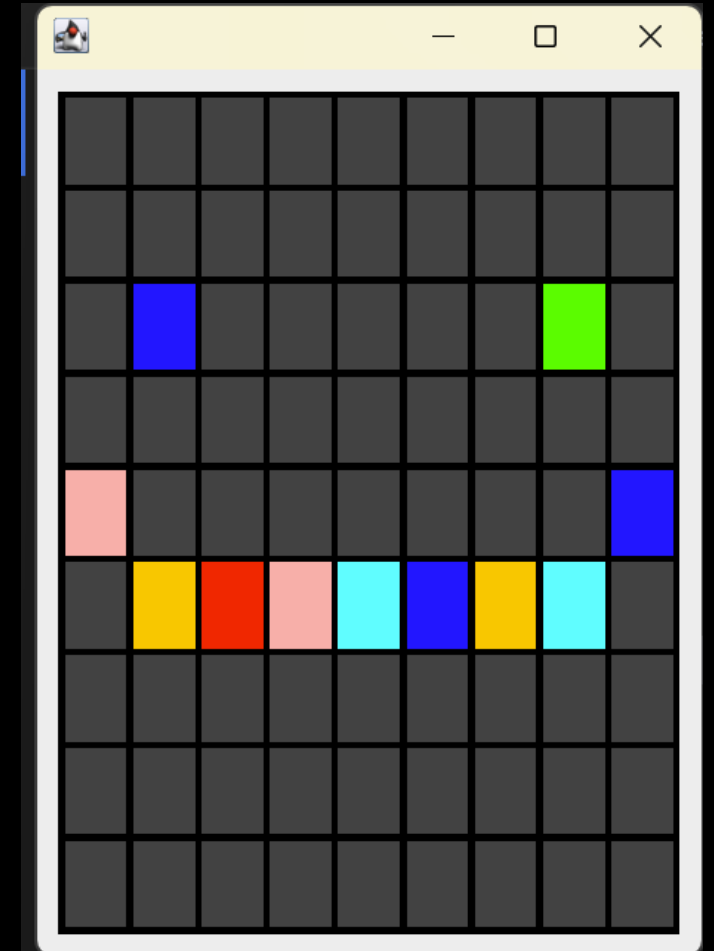
MouseEvent

- Ikke en del av Swing, men java.awt
- En "Listener"-klasse, som er en implementasjon av Observer-mønsteret. Denne observeren sender et signal når komponenten registrerer noe input som oppstår fra museklikk.
- Hvordan dette foregår "under panseret" vil ikke være relevant. Gled dere til INF214 :--)

Enkel GUI-applikasjon: 101-Paint

Læringsplan:

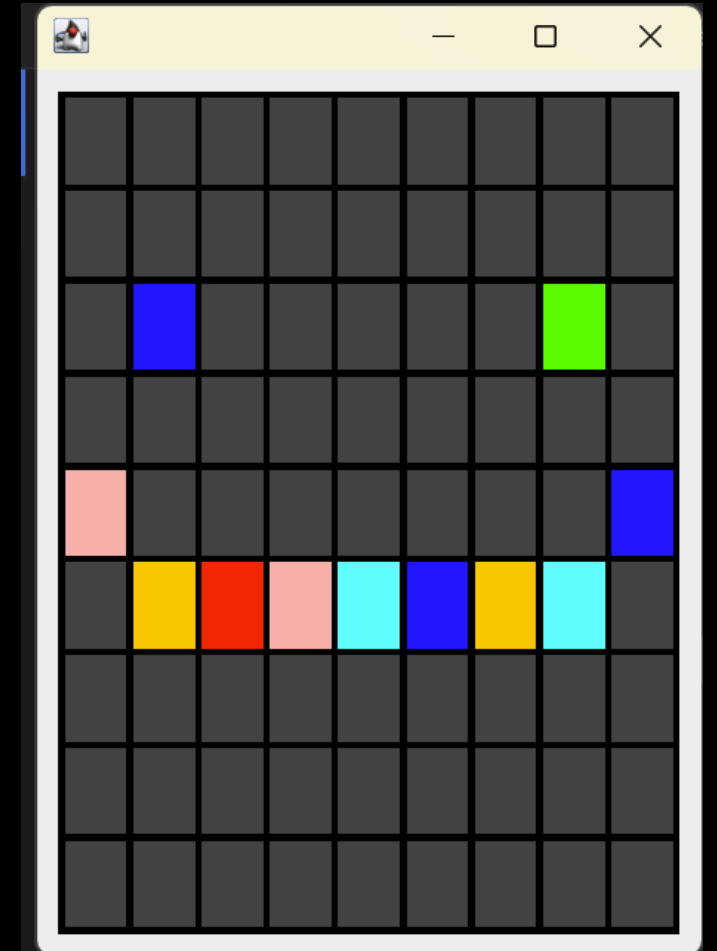
- Demonstrere hvordan MVC kan benyttes for mindre applikasjoner
- Lage en enkel GUI-applikasjon som benytter seg av museklikk.
- Lære oss å gjenbruke kode fra tidligere oppgaver, og tilpasse det etter behov.



Enkel GUI-applikasjon: 101-Paint

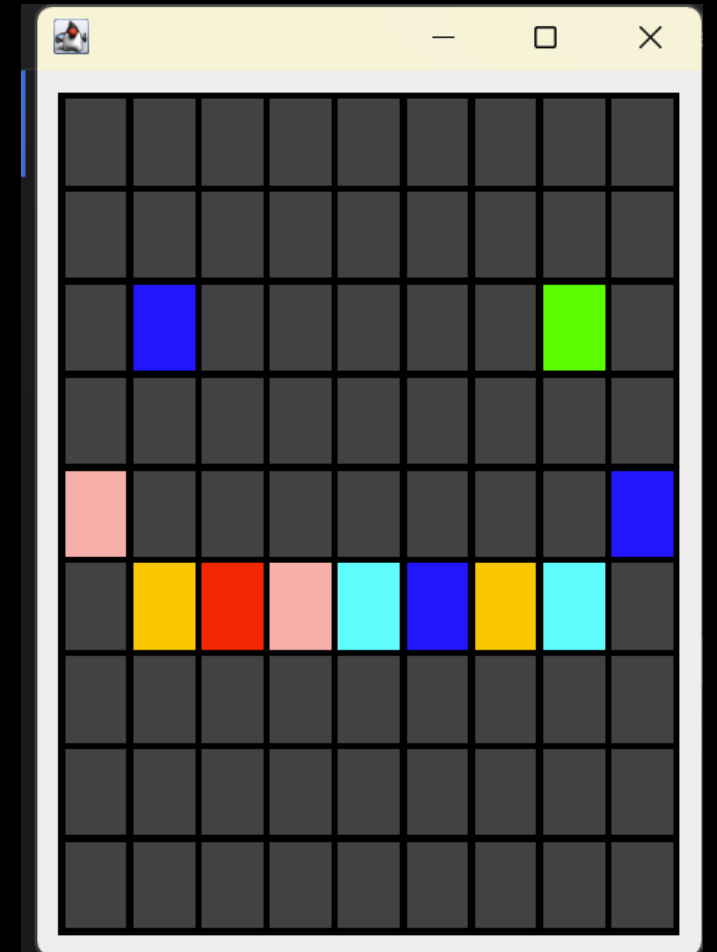
Ønsket resultat:

- Å kunne fargelegge ruter i en Grid med en tilfeldig farge når vi trykker på dem.



Enkel GUI-applikasjon: 101-Paint

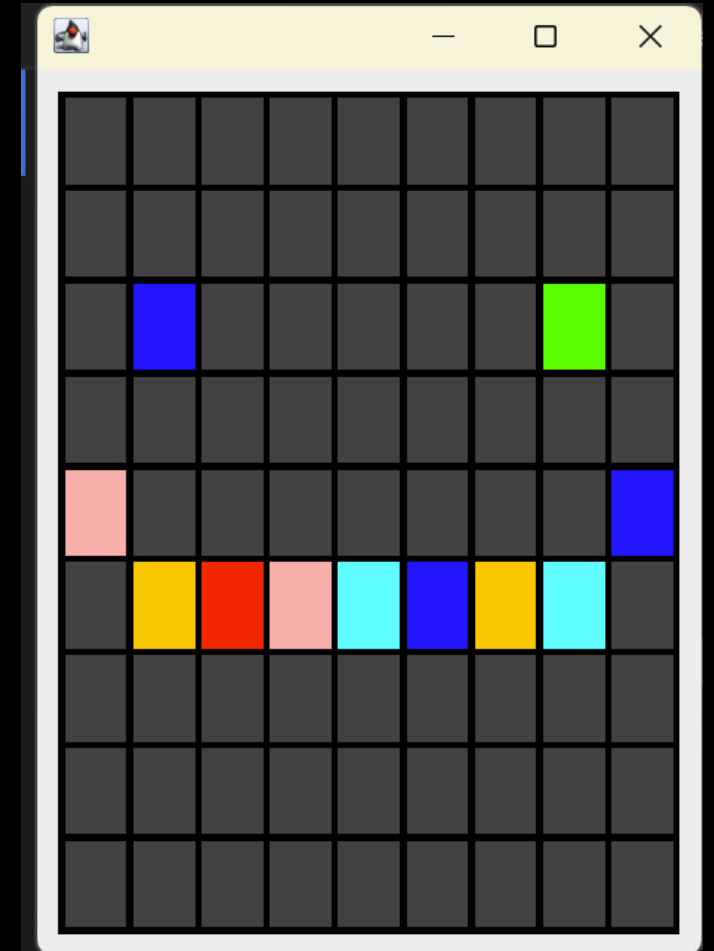
<https://git.app.uib.no/Gisle.Kvamme/crashPaint>



Enkel GUI-applikasjon: 101-Paint

Steg 0: Start

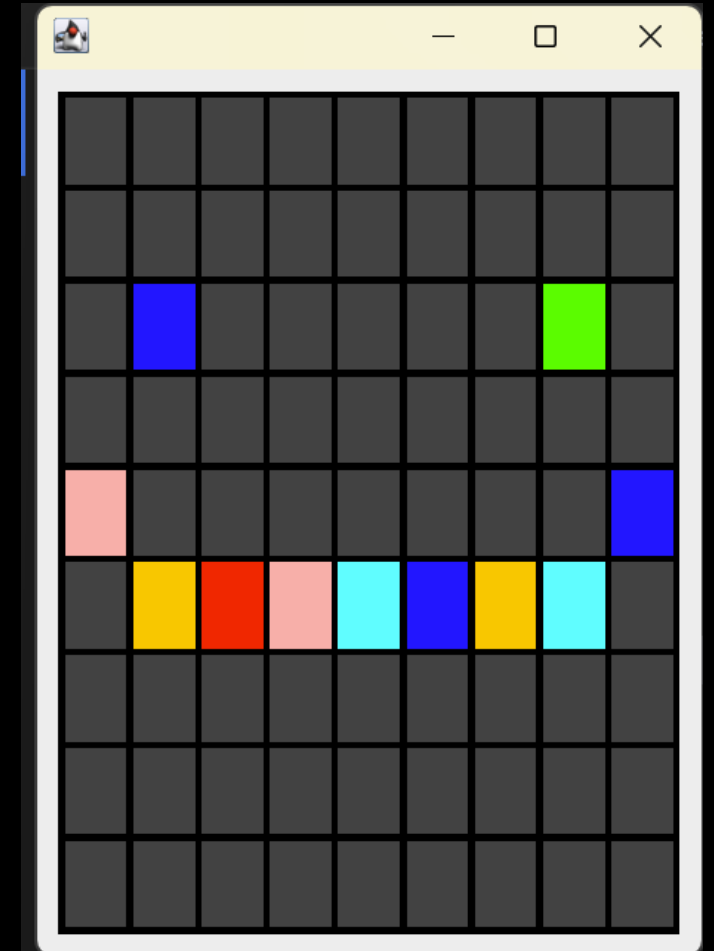
- Se kjapt gjennom filene, og bli litt kjent med koden. Mye av dette burde være ganske kjent for dere allerede, men noe av det er lagt til...
- Se spesielt på `CellPositionToPixelConverter` og `View`



Enkel GUI-applikasjon: 101-Paint

Steg 1: Modellen

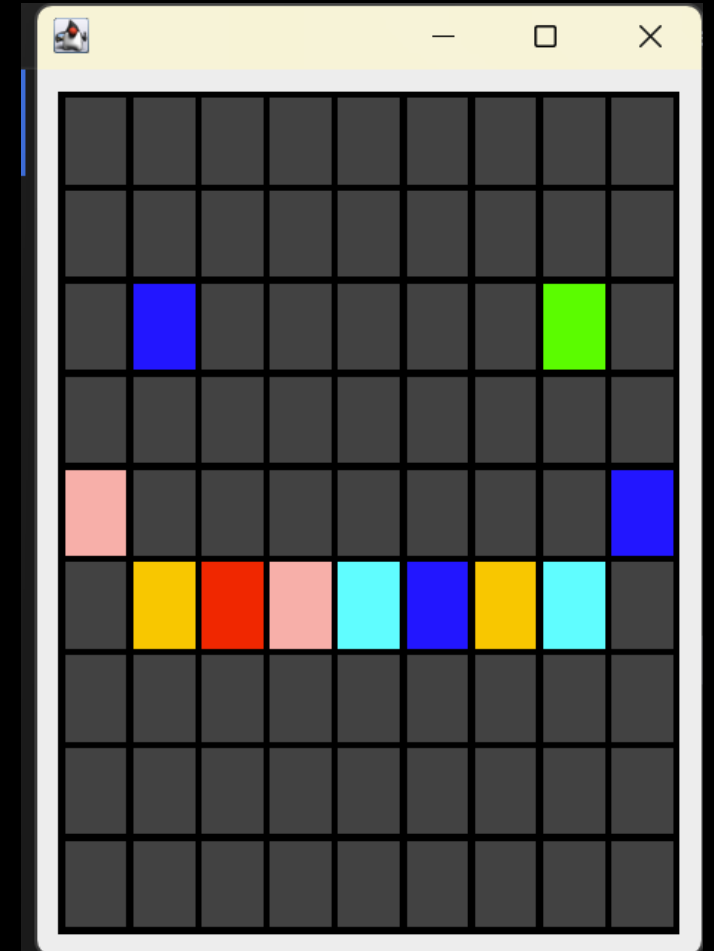
- Lag klassen `RandomColorFactory`, som implementerer `ColorFactory`. Denne klassen er en fabrikk som "lager" en villkårlig farge.
- La Model utvide `Grid` med parametertypen `Color`. Her vil du trenge å kalle på `Grid` sin konstruktør! NB! `Grid` sin konstruktør er en "superkonstruktør"
- Opprett en feltvariabel i modellen, som er en instans av `RandomColorFactory`.



Enkel GUI-applikasjon: 101-Paint

Steg 2: Knyt sammen modell og visning

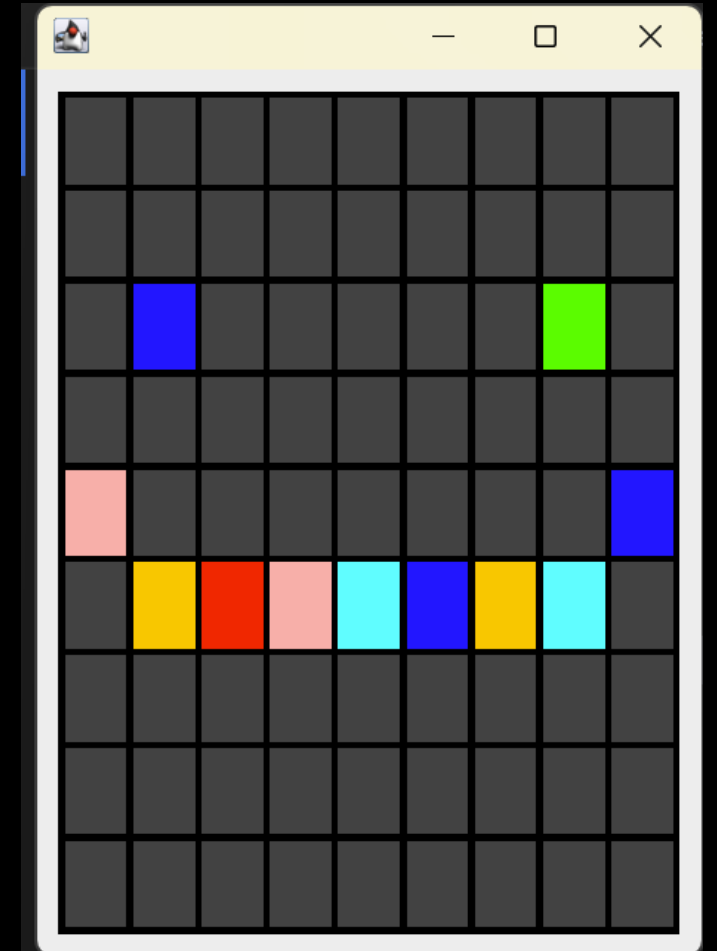
- Gjør det mulig for View å tegne modellen vår ved å knytte klassene Model og View sammen via et restriktivt grensesnitt. NB! Dette grensesnittet finnes allerede.
- Implementer metodene som kreves av grensesnittet. Se på returtypene og prøv å finn den mest konsise implementasjonen.



Enkel GUI-applikasjon: 101-Paint

Steg 3: Vis et rutenett på skjermen

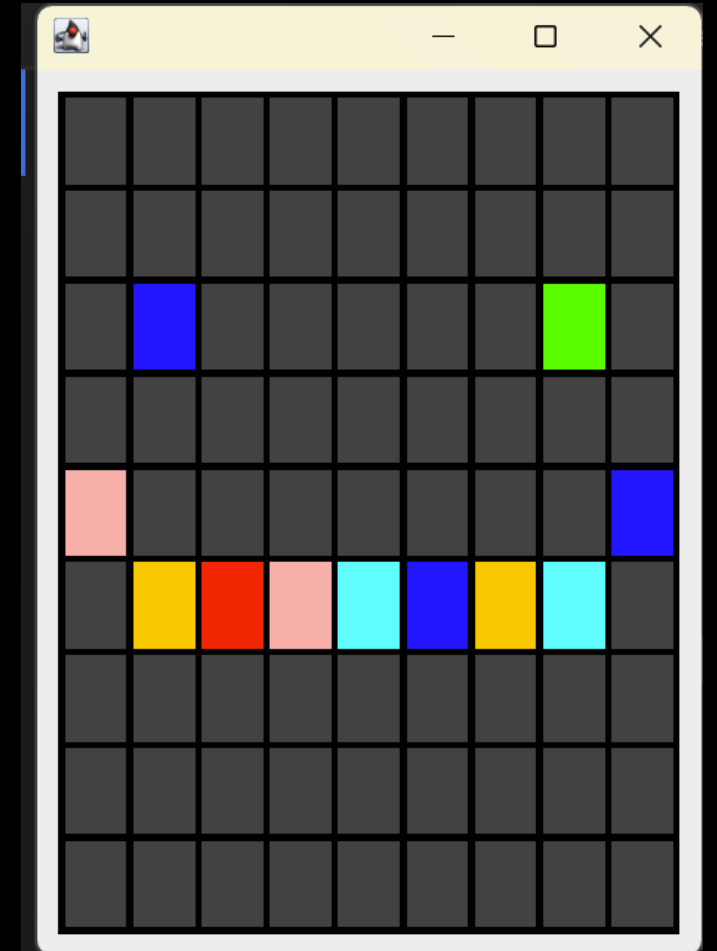
- Gå til Main.java:
- `.setContentPane(...)` tar et JPanel som parameter. Akkurat nå har vi gitt det et tomt panel som argument, hva kan vi gi i stedet, slik at vi kan "vise" en instans av modellen vår?
- Hint: Opprett en instans av modellen. Tenk over hvilke typer denne instansen kan ha.



Enkel GUI-applikasjon: 101-Paint

Steg 4: Controller-pakken

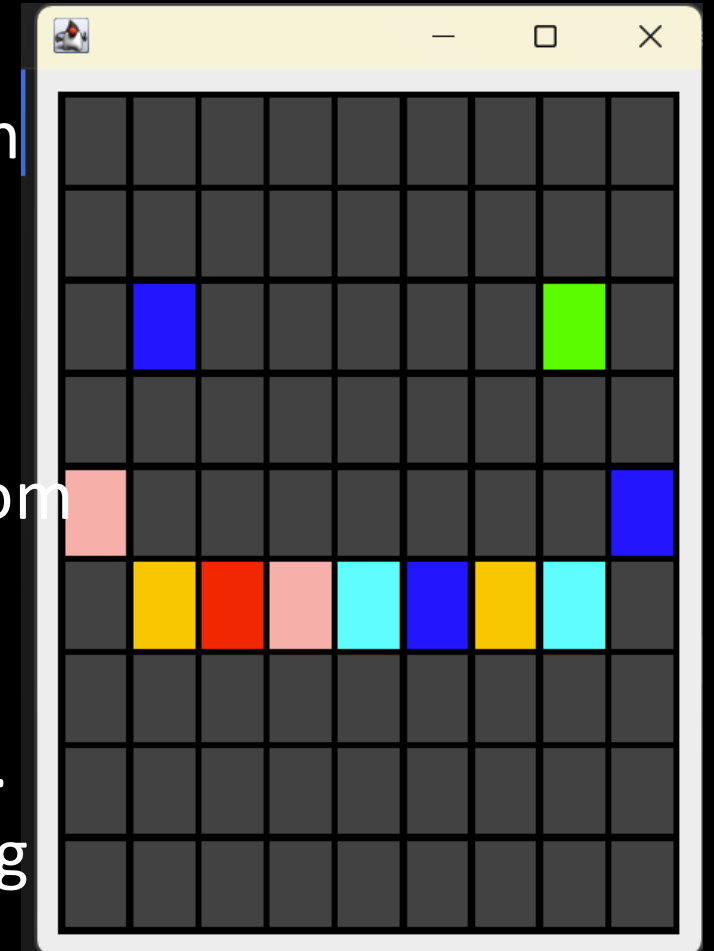
- Opprett pakken "controller" på samme nivå i hierarkiet som "model" og "view".
- Lag et grensesnitt, ControllableModel.java. Hvilke metoder trenger vi for å endre fargen på en rute i modellen vår?



Enkel GUI-applikasjon: 101-Paint

Steg 5: Controller-klassen

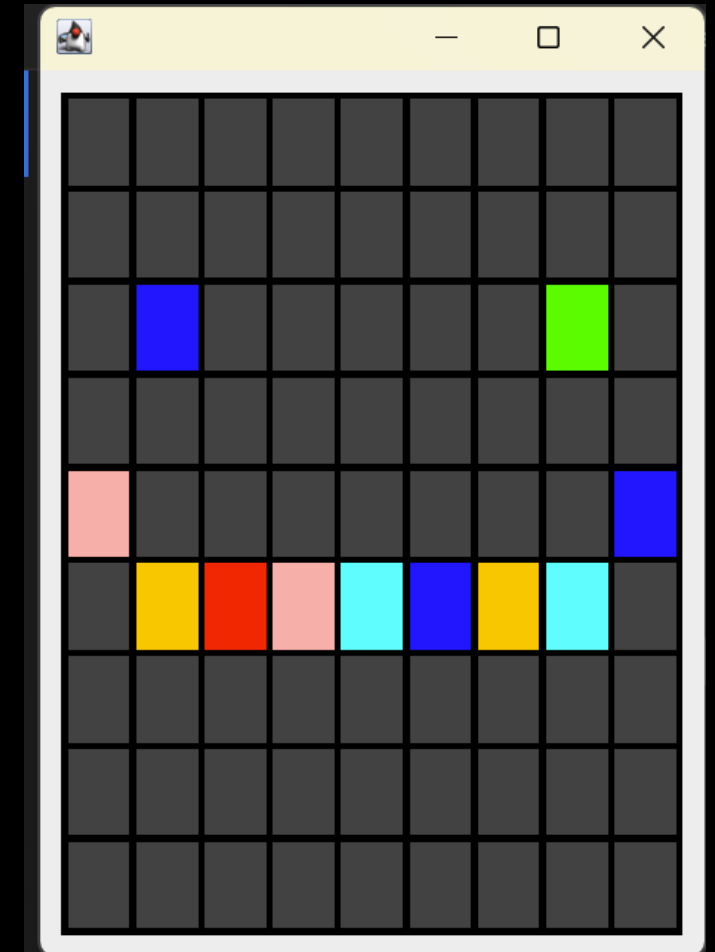
- Opprett klassen `Controller.java`. I konstruktøren, gi controller-klassen restriktiv tilgang på modell-klassen vår. Gi den også tilgang på visningsklassen.
- Kall metoden `setMouseListener` på view-objektet i konstruktøren, gi `"this"` som argument. Hva er det som skjer her?
- Overskriv metoden `"mousePressed(MouseEvent e)"`. Her kan dere få bruk for: `view.getPixelConverter()`, og `e.getPoint()`. NB! `getPoint()`'s returtype er `Point2D`.



Enkel GUI-applikasjon: 101-Paint

Steg 6: Gjør slik at museklikk "fargelegger" ruter

- La modellen implementere `ControllableModell` og implementér nødvendige metoder. Husk å benytt deg av fabrikken du lagde i steg 1.
- Opprett en instans av Controller-klassen i `Main.java`. Du burde nå være i stand til å fargelegge ruter. Hvordan fungerer samhandlingen mellom Controller og View?



Enkel GUI-applikasjon: 101-Paint

Steg 7: Hjemmeoppgave

- Gjør slik at programmet minner litt mer om et vanlig paint-program.

Hint til fremgangsmåte:

- I Main: Øk størrelsen på rutenettet
- I View: Fjern "inner margins" og den grå bakgrunnen
- I Controller: Gjør slik at den tegner kontinuerlig når du holder inne musen, ikke bare på ett og ett klikk. Dette kan oppnås med for eksempel å overskrive `mouseDragged`, eller å bruke `mousePressed` og `mouseReleased` i en løkke.

Enkel GUI-applikasjon: Oppsummering

- Dette var ikke ment for å være en særlig stor programmeringsutfordring, men en teoretisk oppsummering
- Hvis det kommer en GUI-oppgave på eksamen blir programmet til å være større enn dette, men mindre enn en semesteroppgave
- Ytterligere tips: Se på tidligere langsvarsoppgaver og prøv å forstå klassekomposisjonen, og ikke bare "løs" oppgaven
- Lag dine egne små GUI-applikasjoner, gjenbruk kode fra kurset og se hva du kan få til. Lag veldig enkle spill og animasjoner.

Tips til eksamen

- IKKE oppdater OS kvelden før eksamen
- Kan være lurt å ha en alternativ IDE klar
 - VSCode
 - IntelliJ
 - Eclipse
- Sjekk at git og GitLab fungerer som det skal
 - Last ned git (git --version)
 - Koble til GitLab med SSH
- Les mer på uib.no

Generelle tips

- Prøv å unngå stress
- Les gjennom hele oppgaveteksten før du begynner
- Tenk igjennom problemet og legg en liten plan før du koder
- Husk å commit når du er ferdig med en del!
- Ta kortere pauser underveis, gi deg selv tenketid.
- Få mest mulig kode "på papiret", levér heller uferdig kode enn å slette.
- Om du ikke fikk noe helt til, forklar hva du prøvde på.
- Det er bare en eksamen...