INF101 kræsjkurs

Objektorientert programmering

av Alexander Höpner og Gisle Kvamme

Plan for dagen

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

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

```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")
```

```python
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

```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");
}
```

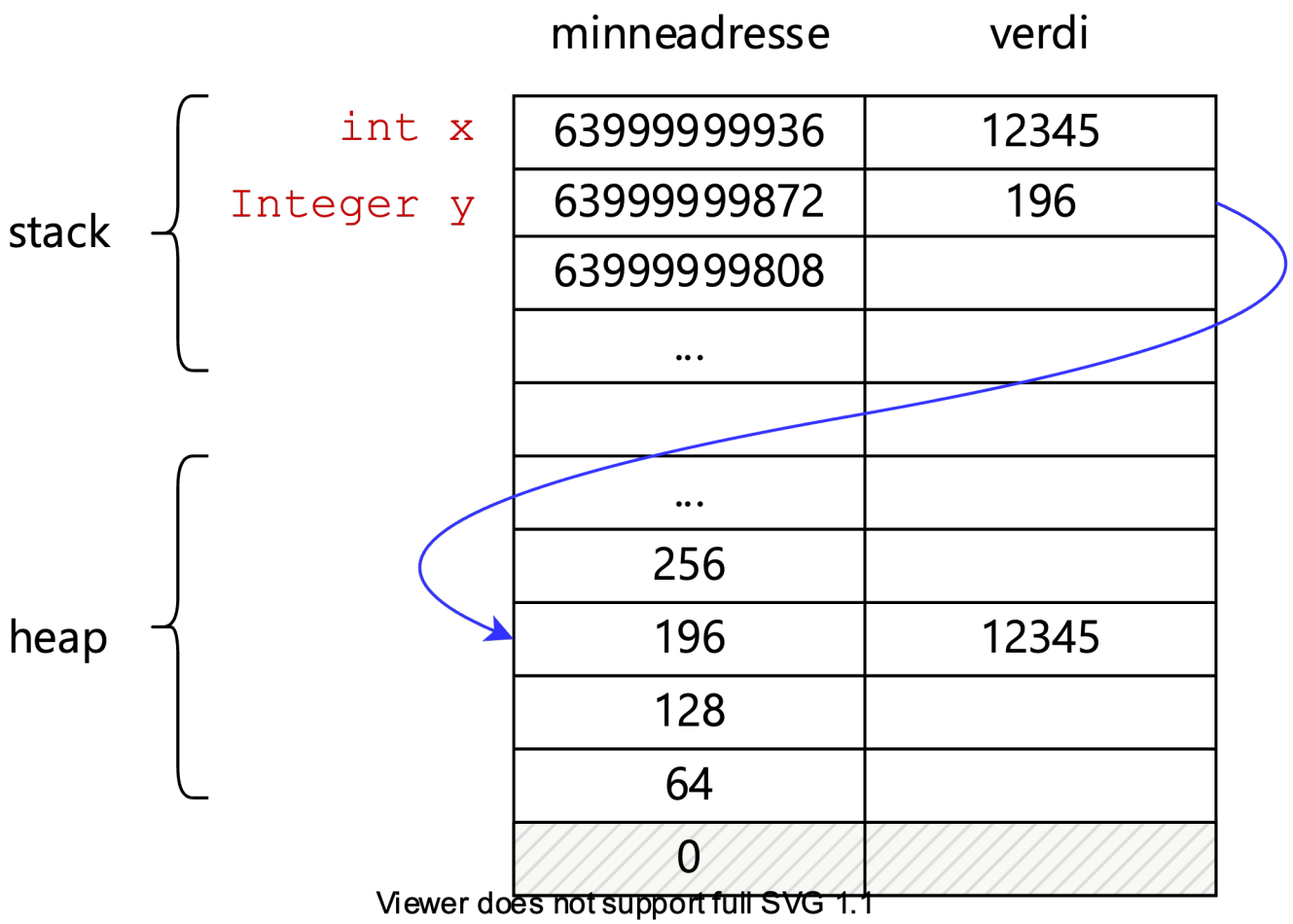
```java
for (int i = 0; i < 10; i++) {
        System.out.println(i);
}

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
heap
int x
Integer y
63999999936
12345
63999999872
196
63999999808
...
...
256
196
12345
128
64
0
Viewer does not support full SVG 1.1

Lister/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 met 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 parametre forventer konstruktøren?

Grensesnitt

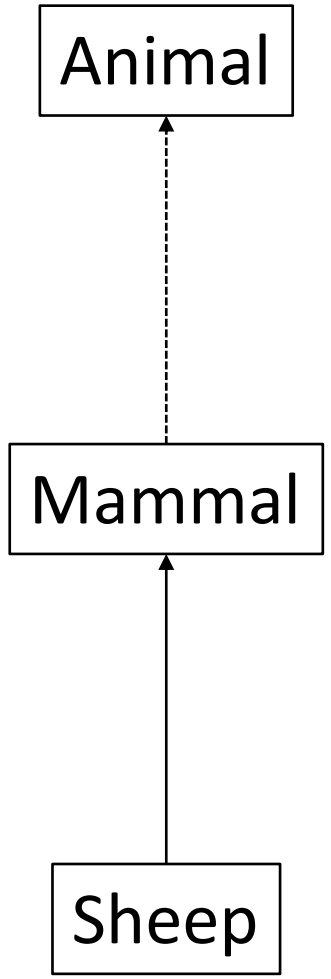
```
public interface Animal {
        String makeSound();
        String getBreed();
        String getName();
}
```

Abstrakte metoder

Arv

```
public abstract class Mammal {

	public boolean laysEggs() {
		return false;
	}
}
```

Arv



Eksamensoppgave V22

```java
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;
	}
}
```

```java
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

```java
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;
	}
}
```

```java
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));
}
```

Equals

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

Java.lang.Object

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

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

```java
/**
* 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 programmering

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
`public`	Ja	Ja	Ja	Ja
`protected`	Ja	Ja	Ja	Nei
~ ingen modifikator ~ (aka «pakke-privat»/standard/default)	Ja	Ja	Nei	Nei
`private`	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

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!"); //!
	}
}
```

```java
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;
    }

}
```

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

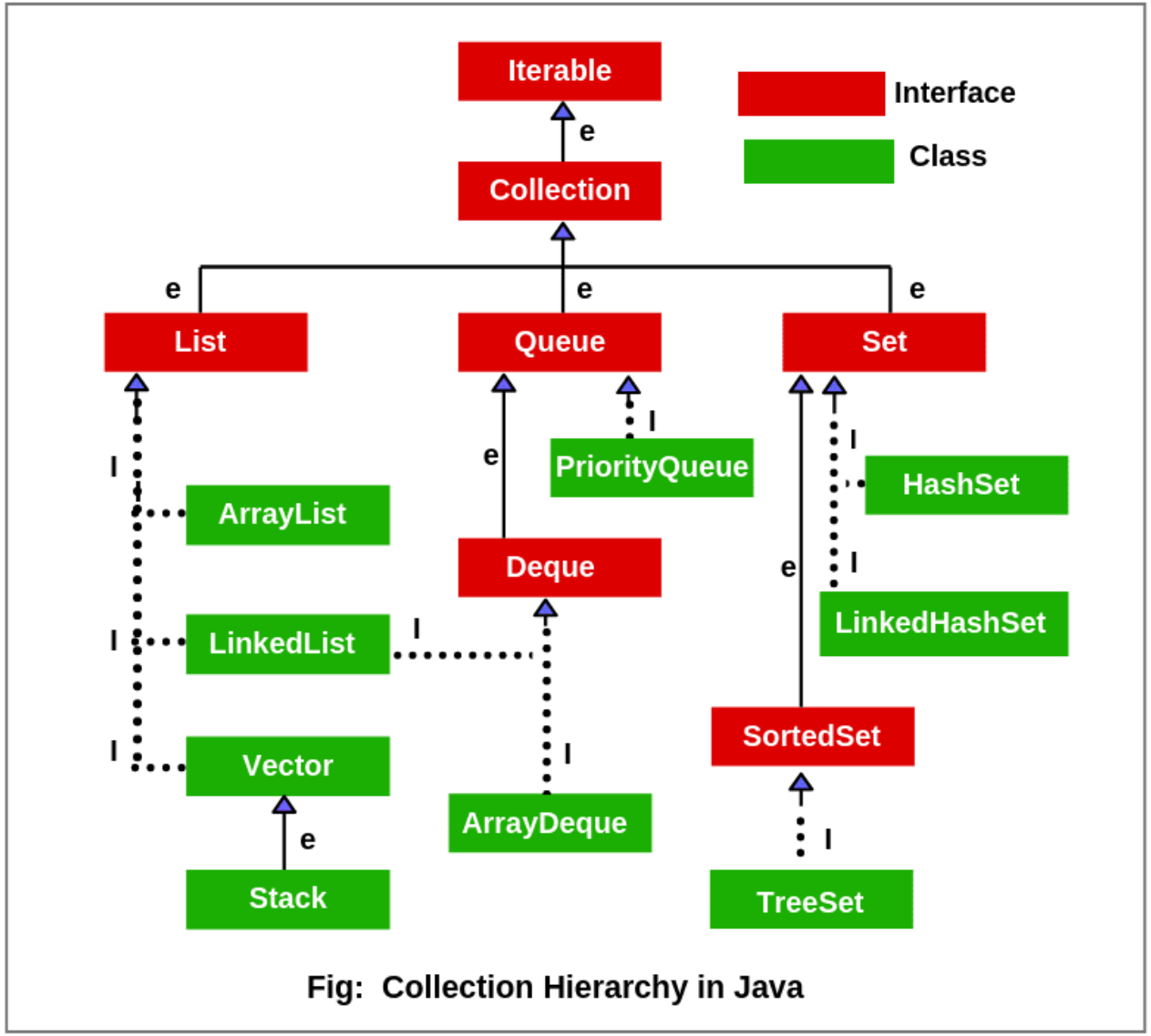


Fig: Collection Hierarchy in Java

Hentet fra [Scientech Easy]

Eksamensoppgave V22

```java
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);
	}
}
```

```java
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

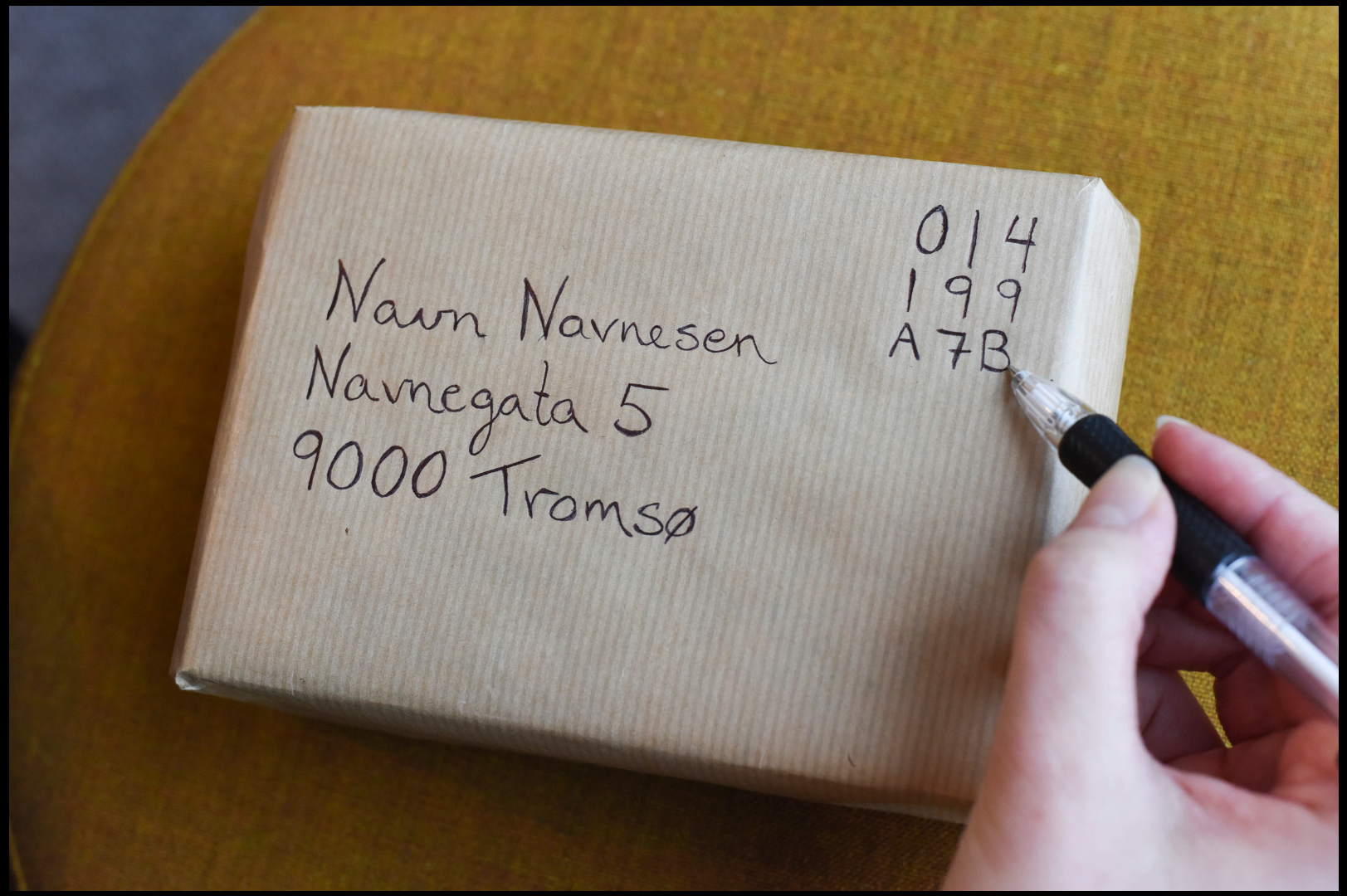
Type	Eksempler
Creational	Factory, singleton, builder
Structural (ikke pensum)	Adapter, decorator, proxy
Behavioral	Iterator, observer, memento, eventbus

Designmønstre II

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

Factory

```
public class ShapeFactory {
    public 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");
        }
    }
}
```



Kilde: posten.no

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);
}
```

```java
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) {}
```

```java
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());
    }
}
```

```java
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

```java
public class Observer<E> {

    private E observedValue;

    public void update(E value) {
        observedValue = value;
    }

    public Object getObservedValue() {
        return observedValue;
    }
}
```

```java
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);
    }
}
```

```java
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
```

Grafikk

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

Swing

- Utvidelse av java.awt, Javas originale bibliotek for GUI, inneholer bl.a. Graphics, støtte for farger, geometriske former.

- Lettere å anvende enn awt

- Platformsuavhengige applikasjoner

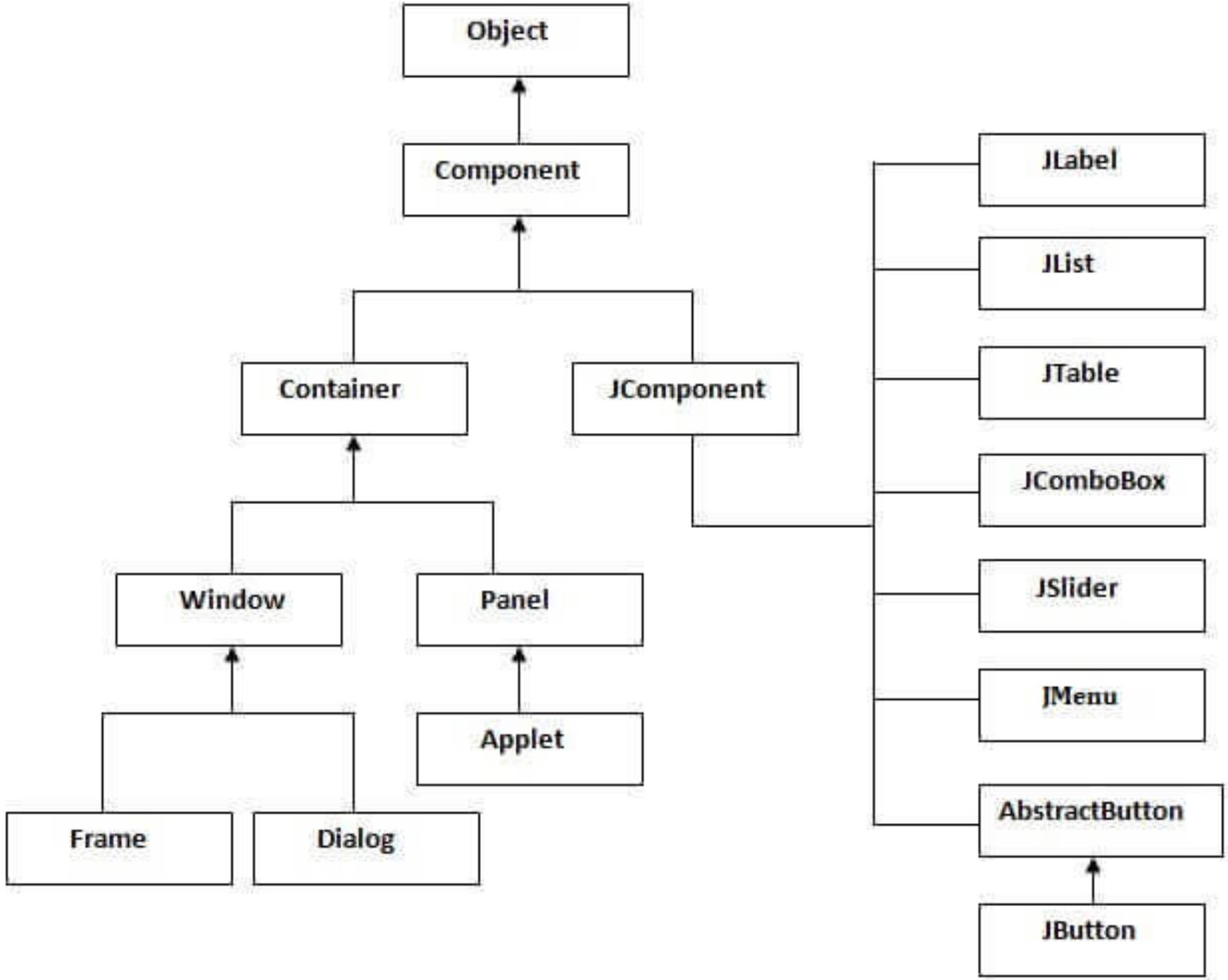
- God støtte for MVC-inndeling

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 seperasjonen gjør det lettere å utvikle i, feilsøke, og utbygge videre på de ulike komponentene


Object
Component
Container
JComponent
Window
Panel
Frame
Dialog
Applet
JLabel
JList
JTable
JComboBox
JSlider
JMenu
AbstractButton
JButton

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 INF113 :--)

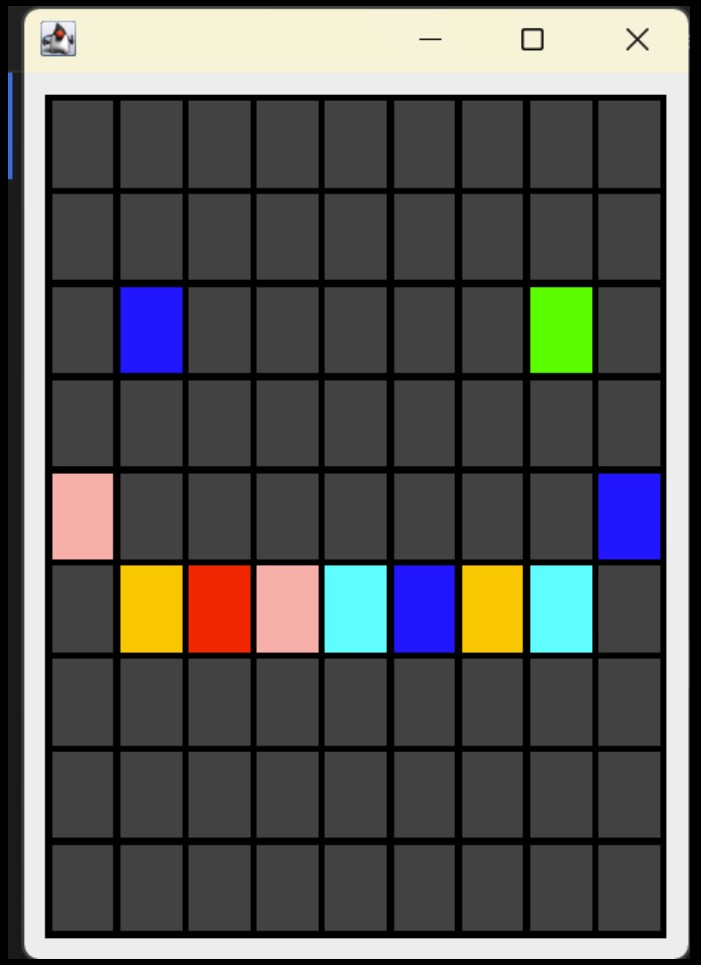
Enkel GUI-applikasjon: 101-Paint

Læringsplan:

- Demonstrere hvordan MVC kan benyttes for mindre applikasjoner

- Lage en enkel GUI-applikasjon som benytter seg av museklikk.

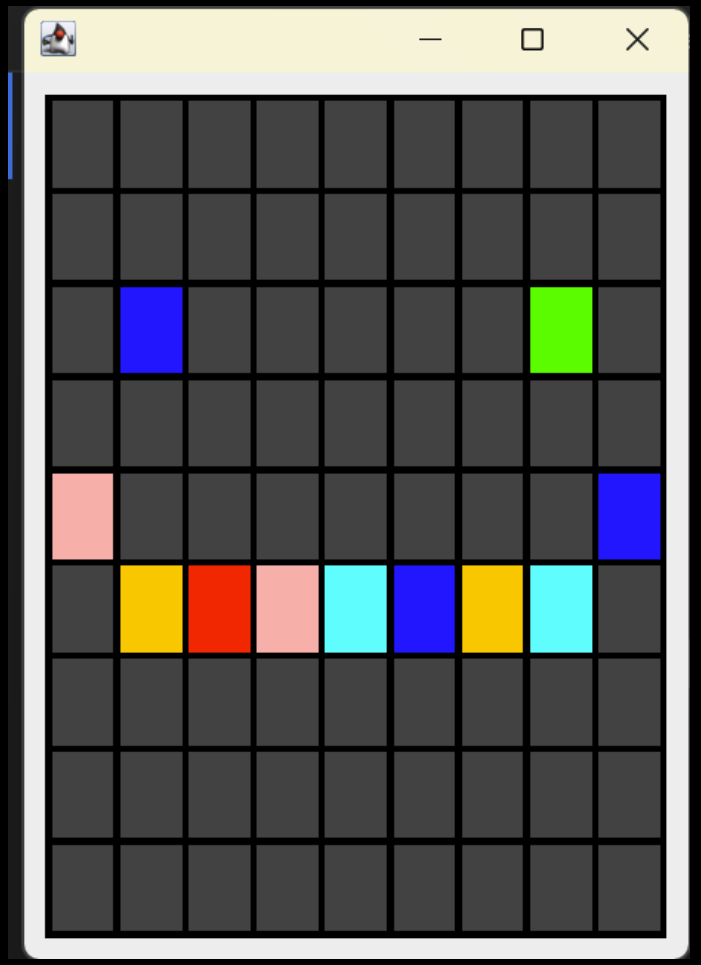
- Lære oss å gjenbruke kode fra tidligere oppgaver, og tilpasse det etter behov.



Enkel GUI-applikasjon: 101-Paint

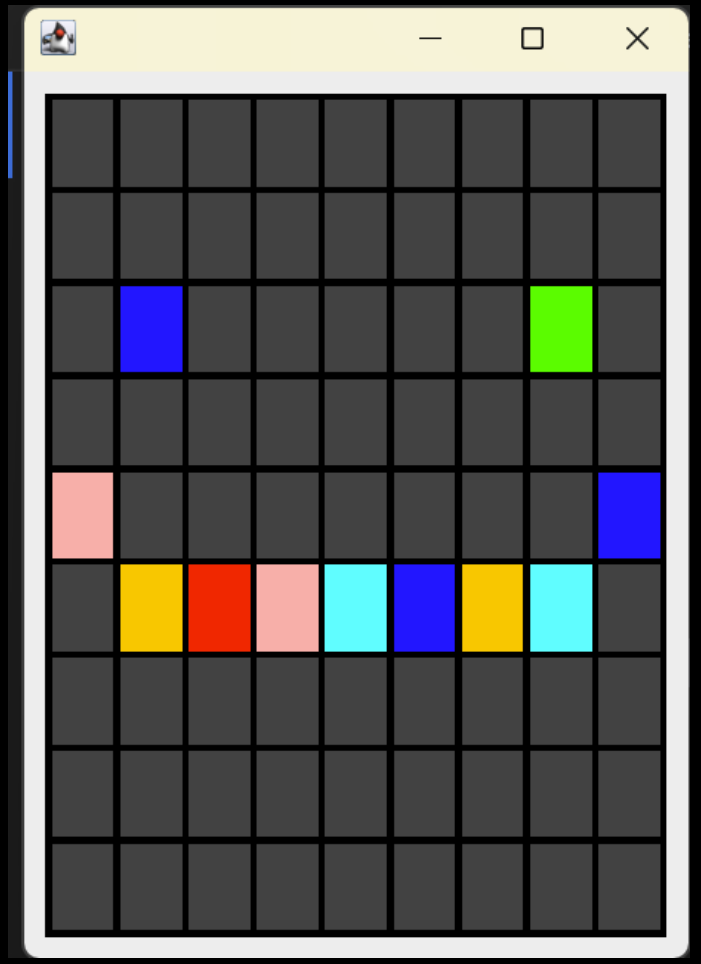
Ønsket resultat:

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



Enkel GUI-applikasjon: 101-Paint

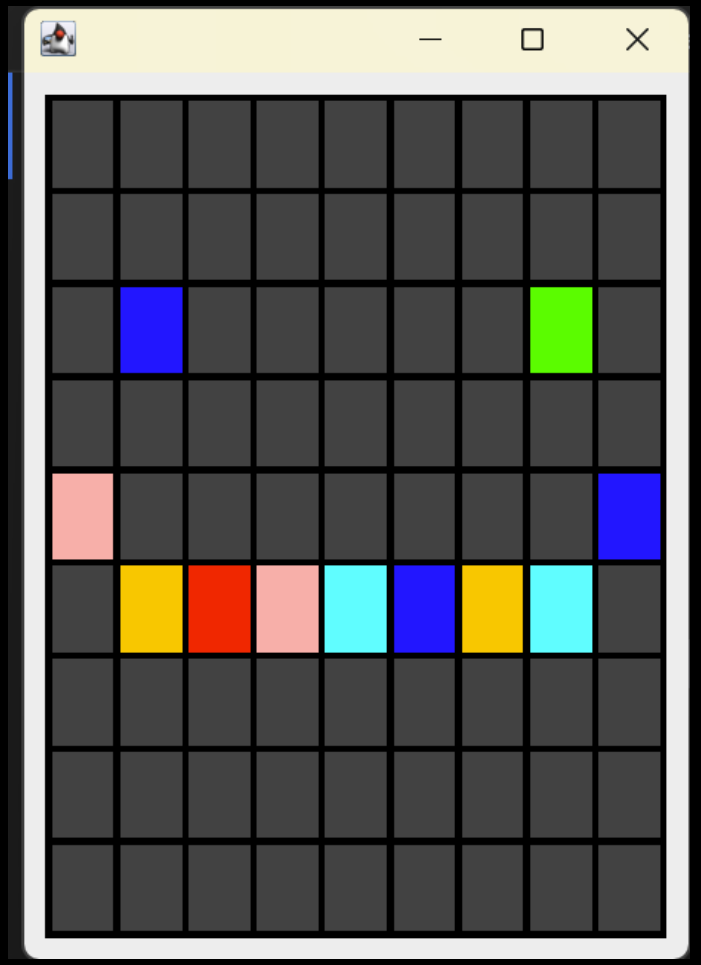
https://git.app.uib.no/Gisle.Kvamme/101PaintStart



Enkel GUI-applikasjon: 101-Paint

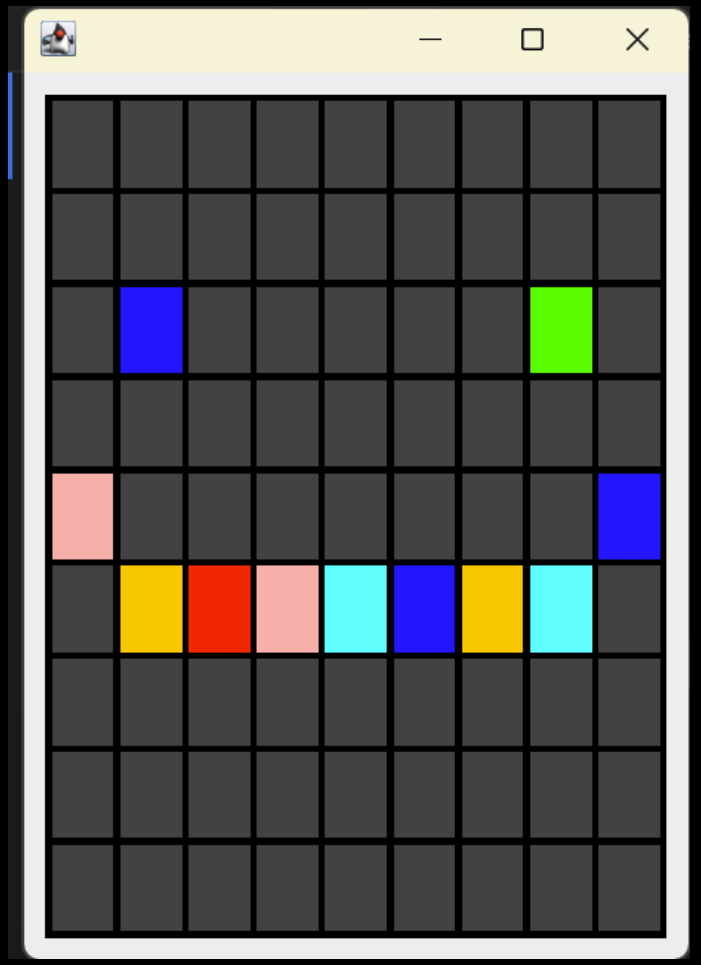
Steg 1: Opprettelse av modellen

- Lag en enkel Factory for den AWT-klassen java.awt.Color som gir en "tilfeldig" farge.
- Oppretter en modellklasse som utvider Grid med parametertypen java.awt.Color.
- Gi modellen en instans av vår Factory.
- Gjør slik at instanser av Model også har typen ViewableModel og ControllableModel.



Enkel GUI-applikasjon: 101-Paint

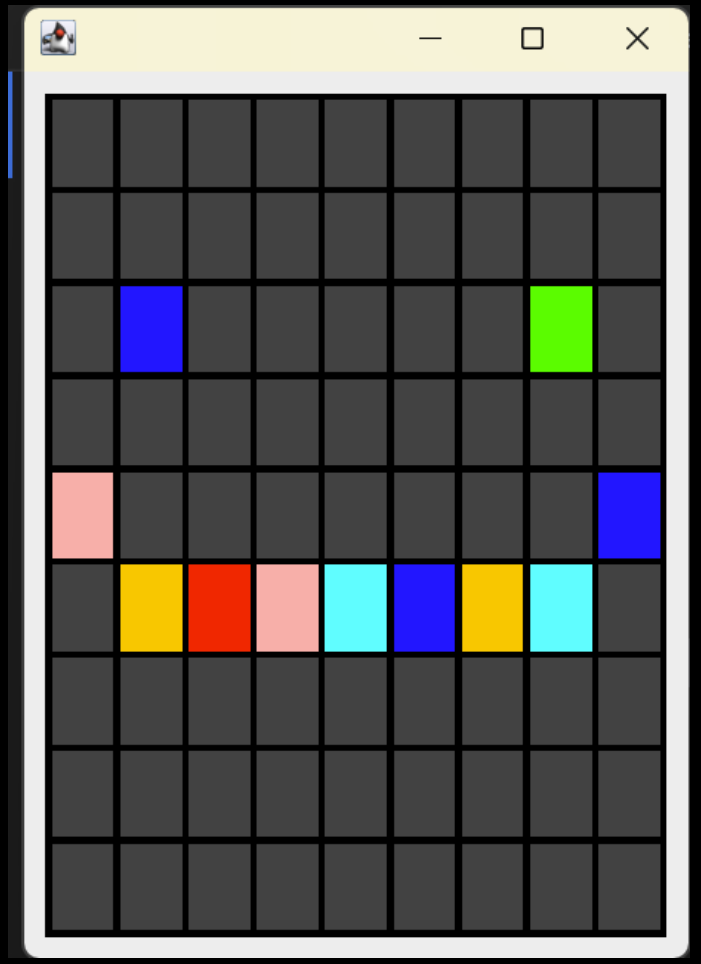
Steg 2: Modifiser visningen fra Tetris for å kunne vise rutenettet vårt



Enkel GUI-applikasjon: 101-Paint

Steg 3: Få opp et rutenett på skjermen

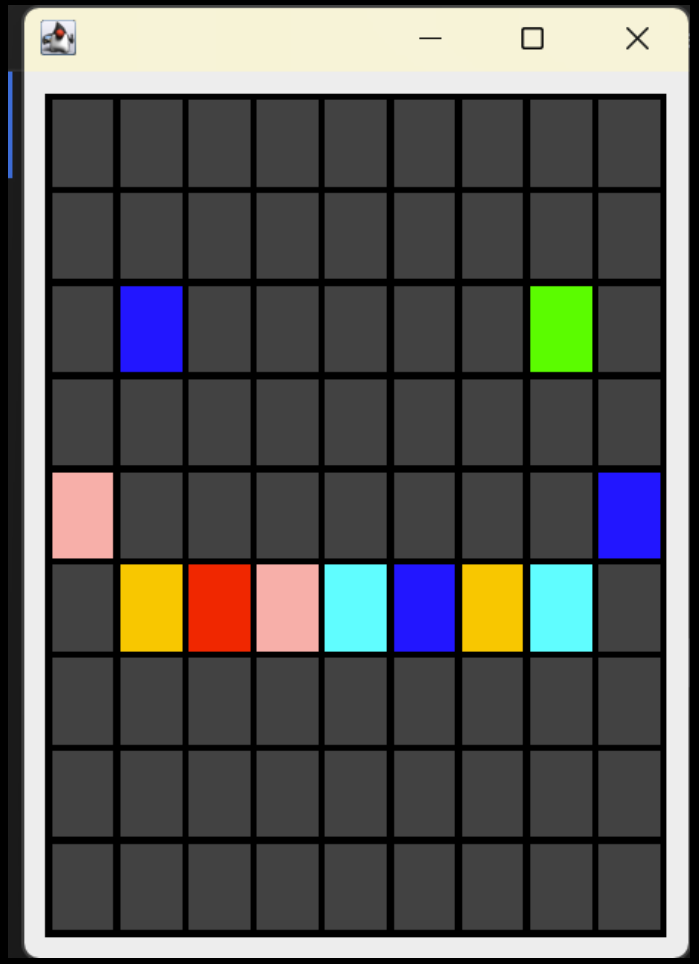
- Kopiér Main-metoden fra Tetris.
- Hvorfor kan kan vi gi en instans av View som argument til opprettelsen av vår JFrame?
- Kjør programmet. Hva ser vi?



Enkel GUI-applikasjon: 101-Paint

Steg 4: Opprett en kontrollørklasse

- La klassen utvide MouseAdapter.
- La klassen ha tilgang på både modellen og visningen.
- Gjør slik at trykk på skjermen kan registreres i visningen.
- Hvilke metoder har vi lyst å overskrive? Hva skal de gjøre?

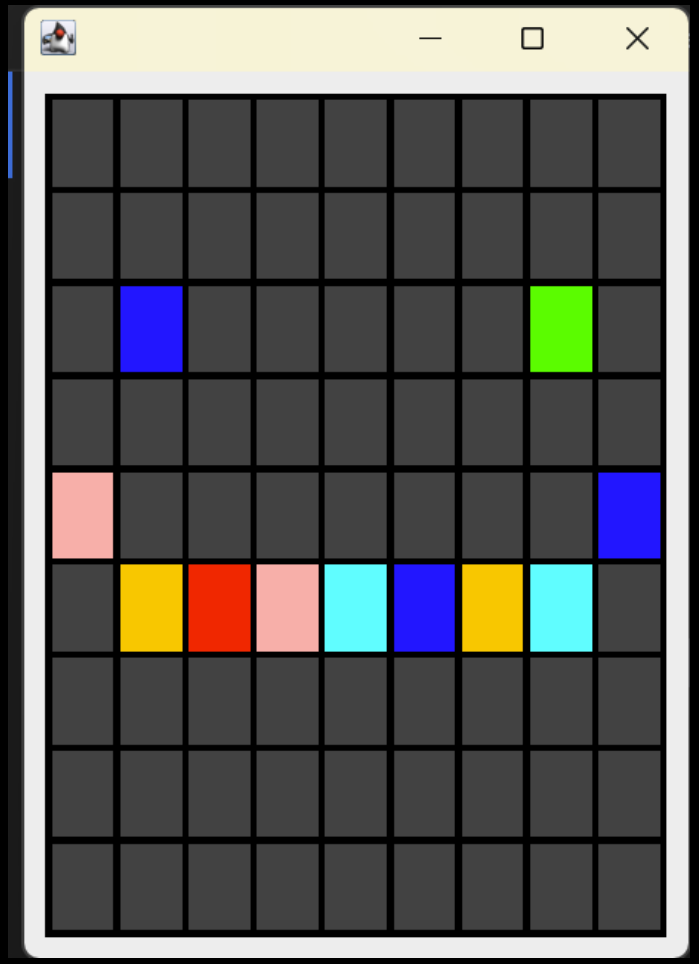


Enkel GUI-applikasjon: 101-Paint

Steg 5: Konverter pekerposisjon til en CellPosition

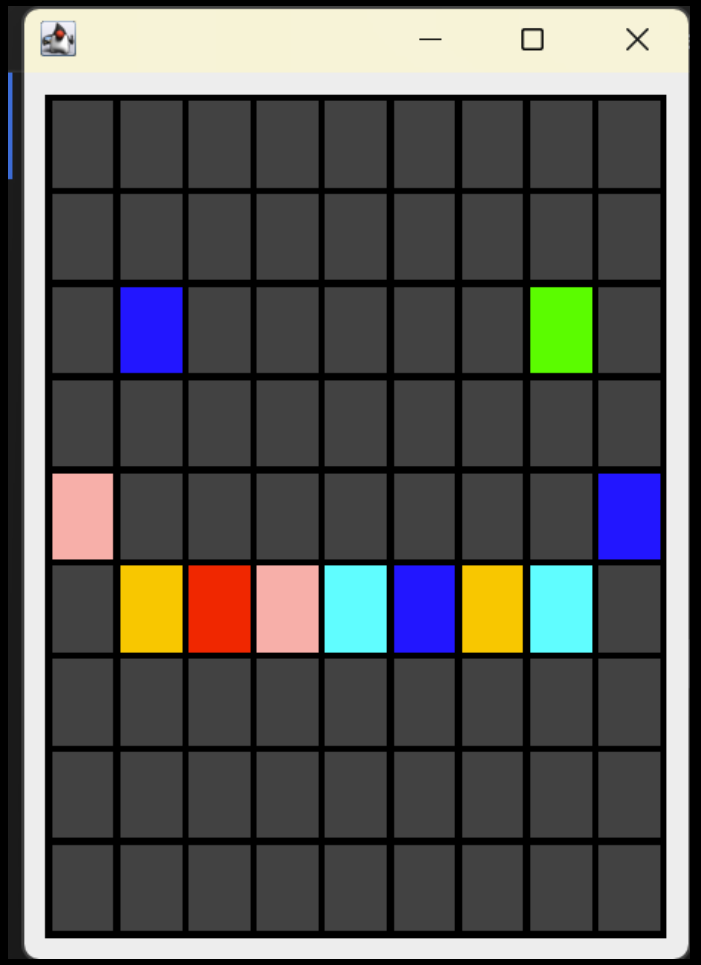
- Hvilken informasjon har vi tilgang på?

- Har vi gjort noe lignende før?



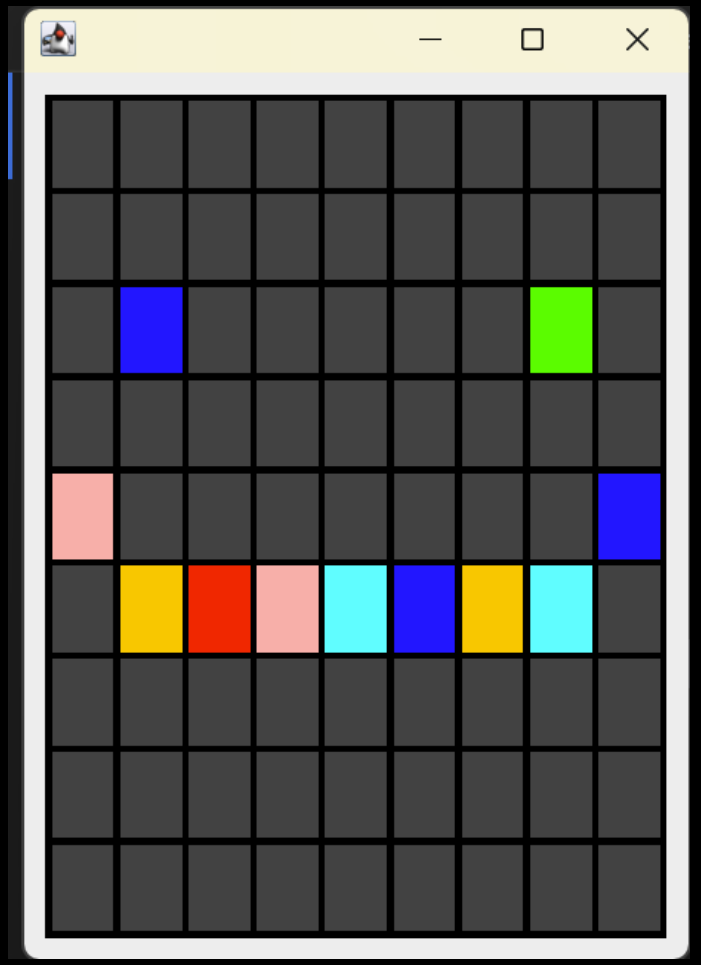
Enkel GUI-applikasjon: 101-Paint

Steg 6: Tilgjengeliggjør visningens CellPositionToPixel-converter ved å modifisere visningen.



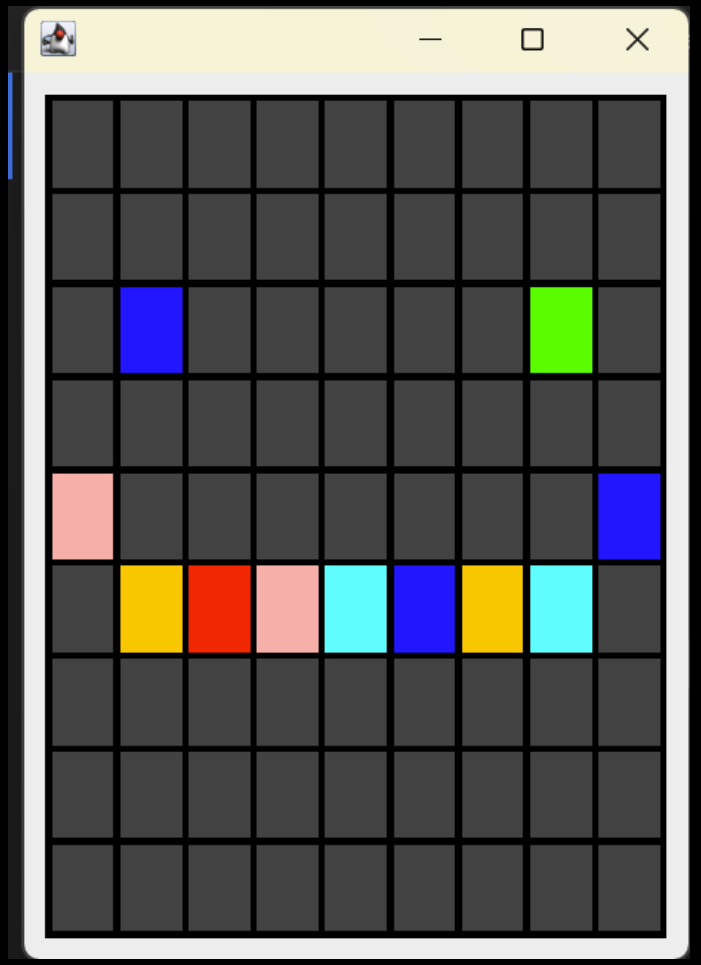
Enkel GUI-applikasjon: 101-Paint

Steg 7: Modifiser CellPositionToPixelConverter,
Slik at vi kan konvertere til en Point til en
CellPosition



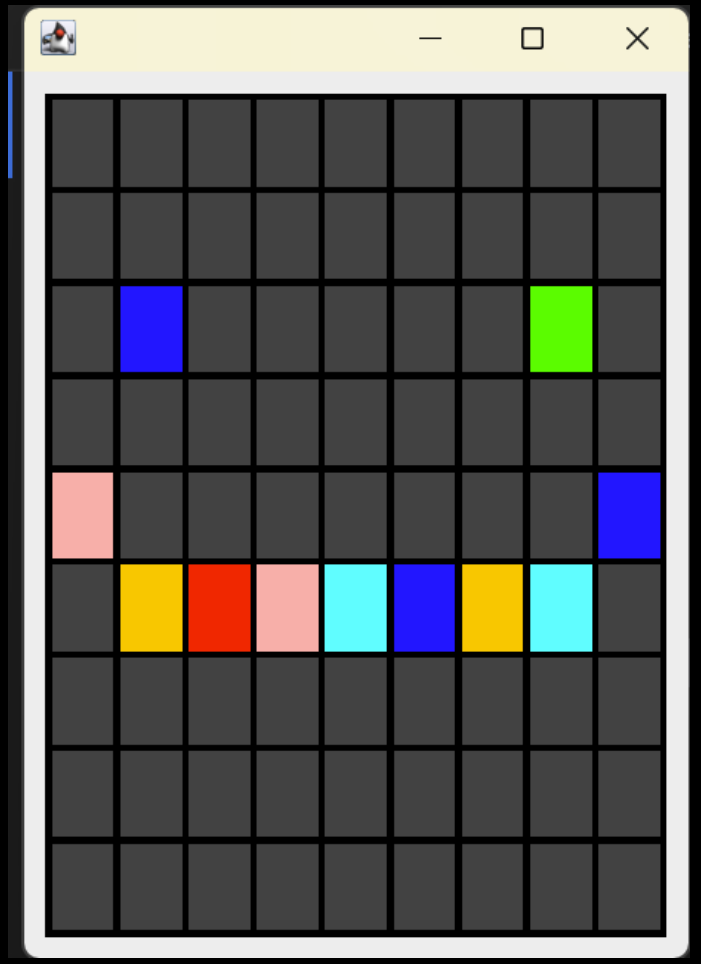
Enkel GUI-applikasjon: 101-Paint

Steg 8: Ferdigstill kontrollør-klassen, og instansier kontrolleren i Main. Kjør programmet.



Enkel GUI-applikasjon: 101-Paint

https://git.app.uib.no/Gisle.Kvamme/101PaintComplete



Eksamen V21

```java
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

```java
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;
```

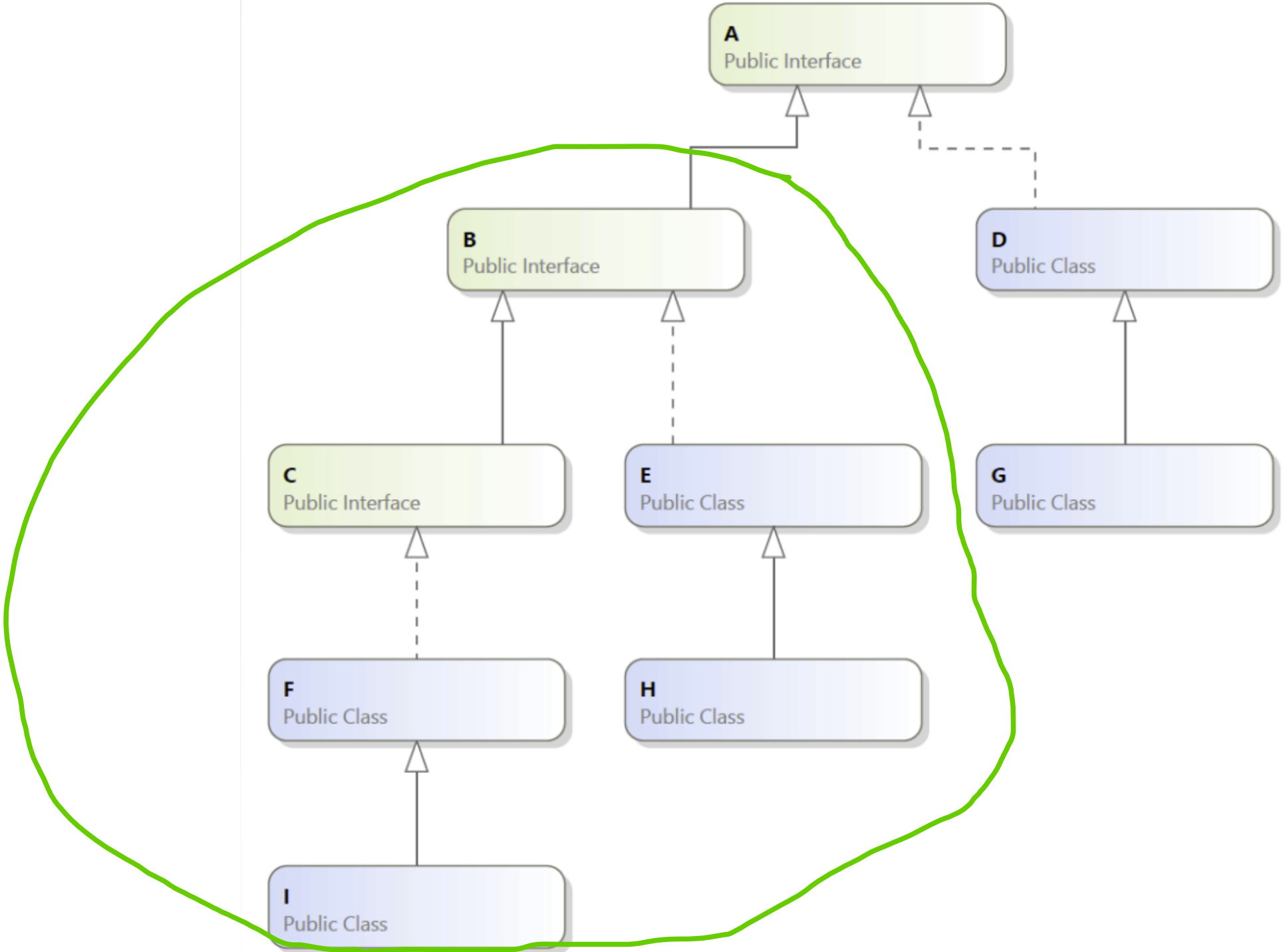
Hvilke typer kan et objekt ha for at det skal kunne legges i variabelen?

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 ✓


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

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

- [x] "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.
- [x] 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.
- [x] Dersom T utvider en klasse som er sammenlignbar med seg selv, så kan List sorteres.

Kodeoppgave: Vaksineplan

Comparable

Du utvikler en programvare som skal fordele vaksiner til pasienter. Hver pasient som skal vurderes har en alder og en *alvorlighetsgrad for underliggende sykdom*. Graden går fra 0, som betyr at pasienten ikke har en underliggende sykdom, til 3, som betyr at pasienten har en alvorlig underliggende sykdom.

Vaksinene kommer på ulike tidspunkt, og programmet må derfor sette opp en vaksineringskø. Hvor en pasient havner i køen avhenger av to (meget forenklede) faktorer:

- Dersom en pasient A har en høyere alvorlighetsgrad enn en pasient B, skal A komme før B i køen.
- Dersom alvorlighetsgraden er lik for to pasienter, skal den eldste av de to pasientene komme først i køen.

1. For å finne ut hvem som trenger vaksinen mest må vi kunne sammenlikne to pasienter. Bruk kriteriene ovenfor til å la Patient implementere Comparable<Patient> slik at den av to pasienter som trenger vaksinen **mest** kommer **før** den andre i en sortert liste.

2. Vaccine-klassen er en spesifikasjon av en generell vaksine. Utvid klassen med to klasser Pfizer og Moderna som henholdsvis representerer vaksinene "Pfizer" og "Moderna". Begge klassene skal ha en konstruktør som tar en LocalDate som argument og setter denne til å være leveringsdatoen.

3. Du skal nå fullføre main-metoden i VaccinePlan-klassen (du kan ta vekk tegnene som kommenterer ut klassen). I main-metoden har du en liste med pasienter og en liste med vaksiner, og du skal tildele vaksinene til pasientene slik at de som trenger vaksinene mest får først. Merk at rekkefølgen i de to listene er tilfeldige. Bruk hjelpemetoden assignVaccine for å tildele en vaksine til en pasient. (Dersom du ikke har fått til de forrige deloppgavene kan du likevel gjøre denne, selv om om du får noen feilmeldinger eller svaret blir feil.)

https://git.app.uib.no/Gisle.Kvamme/crashcoursevaccine

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
  - 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 push 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...