

3.2

```
// Vi skriver en ny klasse Spillbar som abstraherer de felles variablene
// For Lyd og en annen klasse Video
// Vi velger her å gjøre denne abstrakt ettersom det ikke gir mening i
// caset å opprette Spillbar-objekter
public abstract class Spillbar {
    // Flytter de felles variablene til denne klassen
    private String navn;
    private int varighet;
    // Lager en passende konstruktør for de felles variablene.
    public Spillbar(String navn, int varighet) {
        this.navn = navn;
        this.varighet = varighet;
    }
    // Definerer metoden spill som abstrakt ettersom implementasjonen
    // trolig vil være unik for alle underklasser av Spillbar.
    public abstract void spill();
    // Flyter over get-metodene for navn og varighet.
    public String getNavn() {
        return navn;
    }
    public int getVarighet() {
        return varighet;
    }
}

// Vi oppdaterer Lyd-klassen til å nå gjenbruke Spillbar sine egen-
// skaper (Arv).
// Vi må derfor sette Lyd til å arve (extends) fra Spillbar.
public class Lyd extends Spillbar {
    private int kanaler;
    private int maalinger;
    private int bits;
    private LydSamples lydSamples;

    public Lyd(String navn, int varighet, int kanaler, int maalinger,
    int bits, LydSamples lydSamples) {
        // For å sette verdiene for navn og varighet, må vi nå kalle
        // super sin konstruktør i stedet for å sette verdiene direkte.
        super(navn, varighet);

        this.maalinger = maalinger;
        this.bits = bits;
        this.lydSamples = lydSamples;

        if (kanaler >= 1 && kanaler <= 24) {
            this.kanaler = kanaler;
        } else {
            throw IllegalArgumentException;
        }
    }

    public int getKanaler() { return kanaler; }
    public int getMaalinger() { return maalinger; }
    public int getBits() { return bits; }
    public LydSamples getLydSamples() { return lydSamples; }
}
```

```
public class TenHellos {
    public static void main(String[] args) {
        // count from i = 4 to 10
        while (i <= 10) {
            System.out.println(i + "th Hello");
            i = i + 1;
        }
    }
}
```

2.1
Hvilke OOP prinsipper vises i følgende kode?

- Arv og polymorfi
- polymorfi
- * Innkapsling og Arv
- Arv

2.2
Hva blir utskriften fra dette programmet?

- 200
- * 20
- 100
- 1000

2.3
Hva blir utskriften fra dette programmet?

- Vanlig dyr, Vanlig dyr, Vanlig dyr,
- * Vanlig dyr, Honninggrevling, Honninggrevling,

2.4
Hva blir utskriften fra dette programmet?

- [7, 5, 3, 2]
- * [2, 3, 5, 7]
- [2, 5, 7, 3]
- Kompileringsfeil

- 1.1: Konstruktør brukes for å instansiere nye objekter tidligere exam
- 1.2: Abstrakt metode: en metode uten 'kropp' den har bare 'hode'
- 1.3: Instansvariabel kan ikke aksesserer fra static kontekst
- 1.4: private brukes for å innkapsle instansvariabler og metoder
- 1.5: abstrakte klasser kan aldri instansierer
- 1.6: super benyttes hvis du ønsker å kalle en annen konstruktørm metode i samme klasse.
- 1.7: en indre klasse er en klasse definert i en annen klasse.
- 1.8: @Override brukes for reimplementering i en subklasse
- 1.9: En klasse som ikke er abstrakt kan ikke inneholde en abstrakt metode.
- 1.10: Et interface kan inneholde statiske metoder
- 1.11: En klasse i java kan arve fra flere klasser og implementere flere interface
- 1.12: Et objekt beskrives som en instans av en klasse.
- 1.13: A De tre hoveddelene i et UML-klassediagram inneholder klas-senavn, data og metoder.
- 1.14: En klasse som bare har private data og ingen set-metoder er ikke immutable.
- 1.15: A Hvis en metode i en klasse har samme navn, men forskjellige parametere kalles det overloading.
- 1.16: en subklasse er en superklasse
- 1.17: Controller i javafx er vanligvis knyttet til en scene.
- 1.18: i oop illustrerer Arv best gjenbruk av kode.
- 1.19: B double kan holde på det største flyttallet av long, double, byte og float (flyt = desimaltall)
- 1.20: Collections.sort(); må en implementere Comparable-interfacet.

3.5
// Implementerer Comparable for Spillbar slik at underobjekter av denne // klassen kan sammenlignes (basert på navn)
// Vi må definere at det som Spilbar-objekter skal sammenlignes med er // andre objekter av Spillbar (implements Comparable<Spillbar>).
public class Spillbar implements Comparable<Spillbar> {

```
// Fullfører Comparable implementasjonen ved å override compare-
// To-metoden.
// Merk at parameteren er at Spillbar-objekt.
@Override
public int compareTo(Spillbar o) {
    // Klassen String har også en compareTo-metode vi kan benytte for å
    // sammenligne to String-verdier.
    return this.navn.compareTo(o.navn);
}
```

OPPGAVE 2.1

```
public class Sykkel {
    private String type;
    private int gir;
    public Sykkel(String type, int gir) {
        this.type = type;
        this.gir = gir;
    }
}

public class TerrengSykkel extends Sykkel {
    private int seteHoyde;
    public TerrengSykkel(String type, int gir,
    int seteHoyde) {
        super(type, gir);
        this.seteHoyde = seteHoyde;
    }
    public int getSeteHoyde() {
        return seteHoyde;
    }
}
```

OPPGAVE 2.2

```
public class Kalkulator {
    private int tall = 100;
    public void kalkuler(int tall) {
        this.tall = tall * 10;
    }

    private void printTall() {
        System.out.println(tall);
    }
}

public static void main(String[] args) {
    Kalkulator kalkulator = new Kalkulator();
    kalkulator.kalkuler(2);
    kalkulator.printTall();
}
```

OPPGAVE 2.3

```
public class Dyr {
    public void printInfo() {
        System.out.print("Vanlig dyr, ");
    }
}

public class HonningGrevling extends Dyr {
    public void printInfo() {
        System.out.print("Honninggrevling, ");
    }
}

public static void main(String[] args) {
    Dyr dyr1 = new Dyr();
    Dyr dyr2 = new HonningGrevling();
    HonningGrevling dyr3 = new HonningGrev-
    ling();
    dyr1.printInfo();
    dyr2.printInfo();
    dyr3.printInfo();
}
```

OPPGAVE 2.4

```
public static void main(String[] args) {
    List<Integer> tallListe = new Array-
    List<>();
    tallListe.add(2);
    tallListe.add(5);
    tallListe.add(7);
    tallListe.add(3);
    Collections.sort(tallListe);
    System.out.println(tallListe);
}
```