INF101 eksamen vår 2022
Løsningsforslag med sensorveiledning

Torstein Strømme

1 Konstruktør

Automatisk rettet. 1 poeng.

```
public class AIPlayer {

    private String name = "AI Player";

    public void talk() {
        System.out.println("I am a robot brrrr");
    }

    @Override
    public String toString() {
        return name;
    }
}
```

Når man oppretter et nytt objekt av typen AIPlayer, hvor mange parametere forventer konstruktøren?

Riktig svar: 0

2 Person.java

Automatisk rettet. 4 poeng, 0.5 poeng for hvert riktig svar.

```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?

Abstraksjon	Ja
Komposisjon	Ja
Generiske typer	Nei
Instansmetoder	Ja
Feltvariabler	Ja
Polymorfisme	Nei
Innkapsling	Ja
Uforanderlighet (immutability)	Ja

3 Likhet

Forklaringsoppgave. 10 poeng.

```
public class Main {
    public static void main(String[] args) {
        Person x = new Person("Tor", 1999);
        Person y = new Person("To" + "r", 1999);
        System.out.println(x == y);
    }
}
```

Når koden over blir kjørt, printes "false" i konsollen, selv om den som skrev koden hadde forventet å få "true" fordi objektene tross alt er like. Person-klassen er den samme som i forrige spørsmål. Forklar:

- hvorfor svaret blir false, og
- hva som må gjøres for å sammenligne objekter på en god måte, og
- beskriv med ord hvordan man må endre koden for å oppnå dette.

Skriv ca. 3 avsnitt, ikke mer enn 400 ord.

Løsningsforslag:

Når man sammenligner med == sammenlignes den verdien som er på stack'en. For alle refererte typer (i.e. objekter) sammenlignes derfor objektets minneadresse. I dette tilfellet opprettes det to objekter, som vil befinne seg på ulike steder i minne, og x og y er derfor ulike.

Når man skal sammenligne objekter bør man benytte .equals-metoden på objektet, eller eventuelt Objects.equals -metoden fra Java sitt standard-bibliotek.

Det vil likevel ikke være nok å benytte en equals-metode for sammenligning dersom metoden ikke også er implementert/overskrevet i klassen til objektene som skal sammenlignes; den equals-metoden som er arvet fra Object og som benyttes som standard, bruker bare == selv. For å implementere equals-metoden slik at en sammenligning av Person skal gi true i eksempelet over, må .equals overskrives. Den bør sjekke om både navn og fødselsår er like, og returnere true bare hvis begge er det. Sammenligningen av navn må selv benytte seg av .equals, siden String er en referert type.

Sensorveiledning

- 3 poeng: Objektenes minne-adresse sammenlignes eller tilsvarende forklaring
- 2 poeng: equals eller Objects.equals må brukes når sammenligningen skal utføres
- 2 poeng: equals må implementeres/overskrives
- 3 poeng: Skjønn. Demonstrerer kandidaten at hen forstår hva spørsmålet dreier seg om? Det skal ikke trekkes for feil som er primært språklige (som skrivefeil eller grammatikk) men fokus skal være på hvorvidt kandidaten velger gode uttrykk og begreper fra fagfeltet for å beskrive det hen mener.
    - 3 poeng: Bra forklart!
    - 2 poeng: Gir generelt god mening, men blir omstendelig, lite utfyllende, knotete eller litt upresist.

- 1 poeng: Gir noe mening, men er upresist og rotete, bruker feil begreper om ting, eller har klare mangler i resonnementer.
- 0 poeng: Gir liten mening.

4 Bok

Forklaringsoppgave. 10 poeng.

```
public class Main {

    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);
        }
    }
}

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 class Page {

    private static final int MAX_TEXT_SIZE = 3000;
    private String text;

    public Page(String text) {
        if (text.length() > MAX_TEXT_SIZE)
            throw new IllegalArgumentException("Too much text");
        this.text = text;
    }

    public String getText() {
        return text;
    }
}
```

Kodesnutten over har en kompileringsfeil (i main-metoden). Forklar:

- hvorfor denne feilen oppstår, og
- hva man kan gjøre for å fikse den uten å endre noe av koden i main, og
- beskriv med egne ord stegene i løsningen din.

Skriv ca 3 avsnitt, ikke mer enn 400 ord.

Løsningsforslag:

For å benytte en foreach-løkke, må objektet man itererer over implementere grensesnittet Iterable. I dette tilfellet ønsker vi å iterere over Page-objekter samlet i et Book-objekt, så derfor må Book-klassen implementere Iterable<Page>. Siden den ikke gjør det, får vi en kompileringsfeil.

For å løse feilen, kan vi la Book implementere grensesnittet Iterable<Page>. Fordi Iterable er et grensesnitt, må Book da implementere de metodene som Iterable<Page> definerer. Der defineres det en metode som heter iterator uten parametere som returnerer et Iterator<Page> -objekt.

For å implementere metoden er det tilstrekkelig å returnere det samme Iterator-objektet man får ved å kalle på iterator-metoden til feltvariabelen pages. Med andre ord, la iterator-metoden i Book gjøre et kall til iterator-metoden på feltvariabelen pages og returner det Iterator-objektet vi da mottar.

Sensorveiledning

- 3 poeng: foreach-løkker behøver at objektet som itereres over implementerer Iterable
- 2 poeng: den generiske typen brukt i Iterable-grensesnittet må matche den som forventes i løkken (det holder å si at Book må implementere Iterable<Page>)
- 2 poeng: Beskrivelse (la iterator-metoden returnere this.page.iterator())
- 3 poeng: Skjønn. Demonstrerer kandidaten at hen forstår hva spørsmålet dreier seg om? Det skal ikke trekkes for feil som er primært språklige (som skrivefeil eller grammatikk) men fokus skal være på hvorvidt kandidaten velger gode uttrykk og begreper fra fagfeltet for å beskrive det hen mener.
  - 3 poeng: Bra forklart!
  - 2 poeng: Gir generelt god mening, men blir omstendelig, lite utfyllende, knotete eller litt upresist.
  - 1 poeng: Gir noe mening, men er upresist og rotete, bruker feil begreper om ting, eller har klare mangler i resonnementer.
  - 0 poeng: Gir liten mening.

5 Kortbunke

Kodeoppgave. 10 poeng.

I en standard (fransk) kortstokk har hvert kort
- en av 4 mulige «*farger*» (suit): kløver, ruter, hjerter eller spar; og
- en av 13 mulige *valører* (rank): to, tre, fire, fem, seks, sju, åtte, ni, ti, knekt, dronning, konge eller ess.

Det finnes dermed totalt 52 ulike kort.
I repositoriet finner du en klasse *Card* som modellerer et slikt kort.

En *kortbunke* er enkelt og greit en samling av slike kort.

I denne oppgaven skal du implementere en kortbunke på to forskjellig måter. La begge klassene du oppretter være i pakken *inf101v22.cards*.

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 kortbunke. Metoden skal returnere et nytt kortbunke-objekt med alle de 52 forskjellige kortene. Tenk igjennom hvorvidt denne metoden bør være static eller ikke.
- Implementere en metode med signatur *boolean add(Card)* som legger til et kort i kortbunken. 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). Tenk igjennom hvorvidt denne metoden bør være static eller ikke.

Løsningsforslag:

```java
public class InheritedArrayCardPile extends ArrayList<Card> {

    public static InheritedArrayCardPile createFullDeck() {
        InheritedArrayCardPile pile = new InheritedArrayCardPile();
        for (CardSuit suit : CardSuit.ALL_SUITS) {
            for (CardRank rank : CardRank.ALL_RANKS) {
                pile.add(new Card(rank, suit));
            }
        }
        return pile;
    }

    @Override
    public boolean add(Card e) {
        if (e == null) {
            throw new IllegalArgumentException();
        }
        return super.add(e);
```

```
    }
}

public class ComposedArrayCardPile {

    private ArrayList<Card> pile = new ArrayList<>();

    public static ComposedArrayCardPile createFullDeck() {
        ComposedArrayCardPile pile = new ComposedArrayCardPile();
        for (CardSuit suit : CardSuit.ALL_SUITS) {
            for (CardRank rank : CardRank.ALL_RANKS) {
                pile.add(new Card(rank, suit));
            }
        }
        return pile;
    }

    public boolean add(Card e) {
        if (e == null) {
            throw new IllegalArgumentException();
        }
        return this.pile.add(e);
    }
}
```

Sensorveiledning:

- 5 poeng: ComposedArrayCardPile
  - 1 poeng: feltvariabel av typen ArrayList<Card>
  - 1 poeng: add-metoden legger til i feltvariabel
  - 1 poeng: add-metoden kaster IllegalArgumentException
  - 1 poeng: createFullDeck() er static
  - 1 poeng: createFullDeck() har god gjenbruk (bruker en dobbel for-løkke over CardSuit.ALL_SUITS og CardRank.ALL_RANKS)
- 5 poeng: InheritedArrayCardPile
  - 1 poeng: extends ArrayList<Card>
  - 2 poeng: add bruker super-metoden til add (det gis 1 poeng dersom metoden ikke er implementert i det hele tatt, bare arvet)
  - 1 poeng: add kaster IllegalArgumentException
  - 1 poeng: createFullDeck() har god gjenbruk (bruker dobbel for-løkke)

- Dersom løsningene ikke matcher skjema, gjøres en skjønnsmessig vurdering

6 Whac-A-Mole

Kodeoppgave. 35 poeng.

I dette løsningsforslaget utelates oppgaveteksten og løsningene presenteres i kondensert form (uten kommentarer og javadocs etc.). For et fullstendig løsningsforslag, se eget repo.

6a) Oppvarming: restriktive grensesnitt

Kodeoppgave. 4 poeng.

Oppgavesammendrag: Del opp IGrid i GridReadable og GridWritable.
Kondensert løsningsforslag:

```
public interface GridReadable<E> {
    int getRows();
    int getCols();
    E get(Coordinate coordinate);
    boolean coordinateIsOnGrid(Coordinate coordinate);
    Iterable<Coordinate> coordinates();
    Iterable<E> values();
    String toPrettyString(String columnDelimiter, String rowDelimiter);
}

public interface GridWritable<E> extends GridReadable<E> {
    void set(Coordinate coordinate, E value);
    void fill(E value);
    void fill(Function<Coordinate, E> valueProducer);
}

public class ControlledGrid<E> implements GridWritable<E> { ... }
public class ListOfListsGrid <E> implements GridWritable<E> { ... }
```

Ikke nødvendig med endringer i ControlledGrid og ListOfListsGrid utover å endre IGrid til GridWritable.

Oppgavesammendrag: Opprett et restriktivt grensesesnitt GridObservable
Kondensert løsningsforslag:

```
public interface GridObservable<E> extends GridReadable<E> {
    public Observable<E> getObservable(Coordinate coordinate);
}
```

I ControlledGrid:

```
public class ControlledGrid<E> implements GridWritable<E>, GridObservable<E> {
    ...

    @Override
    public Observable<E> getObservable(Coordinate coordinate) {
        ...
    }
```

```
    ...
}
```

Sensorveiledning

- 1 poeng: Refaktorere IGrid til GridWritable
- 1 poeng: GridWritable utvider GridReadable, og metodene er korrekt fordelt mellom de to grensesnittene
- 1 poeng: GridObservable korrekt utført
- 1 poeng: Skjønn. Demonstrerer kandidaten at hen forstår hva hen driver med, eller blir det rot og knot?

6b) Grunnstruktur

15 poeng.

Oppgavesammendrag: Lag Whac-A-Mole ferdig frem til punktet at det fungerer å klikke for å flytte muldvarpen.

```
public interface WhacAMoleViewable {
    GridObservable<Character> getBoard();
}

public class WhacAMoleModel implements WhacAMoleViewable {
    private final static int ROWS = 5;
    private final static int COLS = 7;
    public final static char BLANK = '-';
    public final static char MOLE = 'x';
    private final static Coordinate CENTER = new Coordinate(ROWS / 2, COLS / 2);

    private ControlledGrid<Character> grid = new ControlledGrid<>(ROWS, COLS);
    private Random random = new Random();

    /** Creates a new model for a game of Whac-A-Mole. */
    public WhacAMoleModel() {
        reset();
    }

    /** Reset the game. */
    public void reset() {
        grid.fill(BLANK);
        grid.set(CENTER, MOLE);
    }

    /** Attempt to whac a mole at the given coordinate. */
    public void whac(Coordinate coordinate) {
        if (Objects.equals(MOLE, grid.get(coordinate))) {
            grid.set(coordinate, BLANK);
            grid.set(getRandomCoordinate(), MOLE);
        }
    }

    private Coordinate getRandomCoordinate() {
        int row = random.nextInt(grid.getRows());
        int col = random.nextInt(grid.getCols());
        return new Coordinate(row, col);
    }

    @Override
    public GridObservable<Character> getBoard() {
        return grid;
    }
}
```

```java
public class BoardView extends JComponent {
    private static final int GAP = 2;

    public BoardView(GridObservable<Character> board,
                     WhacAMoleController controller) {
        setLayout(new GridLayout(board.getRows(), board.getCols(), GAP, GAP));
        for (Coordinate coordinate : board.coordinates()) {
            TileView tileView = new TileView(
                board.getObservable(coordinate),
                coordinate,
                controller
            );
            add(tileView, coordinate.row, coordinate.col);
        }
    }
}

public class TileView extends JComponent implements MouseListener {
    private static final int PREFERRED_SIDE_WIDTH = 30;

    private Observable<Character> tileState;
    private Coordinate coordinate;
    private WhacAMoleController controller;

    public TileView(Observable<Character> tileState, Coordinate coordinate,
                    WhacAMoleController controller) {
        this.tileState = tileState;
        this.coordinate = coordinate;
        this.controller = controller;
        tileState.addObserver(this::repaint);
        addMouseListener(this);
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        Color color;
        if (Objects.equals(WhacAMoleModel.MOLE, tileState.getValue())) {
            color = Color.MAGENTA;
        } else {
            color = Color.BLACK;
        }
        g.setColor(color);
        g.fillRect(0, 0, getWidth(), getHeight());
    }

    @Override
    public Dimension preferredSize() {
        return new Dimension(PREFERRED_SIDE_WIDTH, PREFERRED_SIDE_WIDTH);
    }
```

```
    @Override
    public void mousePressed(MouseEvent e) {
        controller.tilePressed(coordinate);
    }

    @Override public void mouseClicked(MouseEvent e) { /* ignore */ }
    @Override public void mouseReleased(MouseEvent e) { /* ignore */ }
    @Override public void mouseEntered(MouseEvent e) { /* ignore */ }
    @Override public void mouseExited(MouseEvent e) { /* ignore */ }
}

public class WhacAMoleView extends JPanel {
    private static final int GAP = 10;

    public WhacAMoleView(WhacAMoleViewable model, WhacAMoleController controller) {
        setLayout(new BorderLayout(GAP, GAP));
        setBorder(new EmptyBorder(GAP, GAP, GAP, GAP));

        add(new HeadUpDisplay(), BorderLayout.NORTH);
        add(new BoardView(model.getBoard(), controller), BorderLayout.CENTER);
        add(new ButtonsPanel(), BorderLayout.SOUTH);
    }
}

public class WhacAMoleController {
    private WhacAMoleModel model;

    public WhacAMoleController(WhacAMoleModel model) {
        this.model = model;
    }

    public void tilePressed(Coordinate coordinate) {
        model.whac(coordinate);
    }
}

public class App {

    public static void main(String[] args) { new App(); }

    public App() {
        JFrame frame = new JFrame("INF101 Whac-A-Mole");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        WhacAMoleModel model = new WhacAMoleModel();
        WhacAMoleController controller = new WhacAMoleController(model);
        WhacAMoleView view = new WhacAMoleView(model, controller);

        frame.setContentPane(view);
        frame.pack();
```

```
        frame.setVisible(true);
    }
}
```

Sensorveiledning for grunnstruktur

Funksjonalitet (10 poeng, tilsvarer ca de grønne rutene over):

Hvis det virker å kjøre programmet, og å klikke på muldvarpen får den til å flytte seg, tildeles automatisk 10 poeng. Dersom brettet vises, men muldvarpen flytter seg ikke, tildeles automatisk 8 poeng.

Dersom det ikke virker, benytt listen under som rettesnor. Kombineres med skjønn.

Modellen (4 poeng)

- 1 poeng: modellen har et grid av Character som feltvariabel
- 1 poeng: modellen har en metode for reset, som blir kalt og flytter muldvarpen til midten.
- 1 poeng: modellen har en metode for å banke en muldvarp som fjerner muldvarpen og oppretter en ny muldvarp på et tilfeldig sted. Metoden sjekker at det faktisk er en muldvarp der det blir banket før prosessen utføres
- 1 poeng: modellen implementerer WhacAMoleViewable, og returnerer brettet i getBoard.

Visningen (4 poeng)

- 1 poeng: visningen får modellen som argument i App uten kompileringsfeil, og typen som brukes i visningen er WhacAMoleViewable
- 1 poeng: layout blir satt riktig i BoardView
- 1 poeng: det blir opprettet riktig antall TileView i BoardView via løkke
- 1 poeng: paintComponent i TileView er korrekt, farge bestemmes på bakgrunn av observerbart objekt sin getValue-metode.

Kontroll (2 poeng)

- 1 poeng: tilePressed -metoden i kontrolleren er korrekt
- 1 poeng: mousePressed -metoden i TileView er korrekt

Generelt (5 poeng)
Disse poengene kan også påvirkes av helheten etter at «forbedringer» er utført. For eksempel dersom det brukes dårlig stil under forbedringer, kan det gi trekk her.

- 1 poeng: konstanter (rows, cols, blank, mole) defineres som konstanter, ikke magiske verdier.
- 1 poeng: gode metodenavn for feltvariabler, parametere og metodene i modellen. Om minst to av dem er dårlige, forsvinner poenget.
- 3 poeng: Skjønn. Demonstrerer kandidaten at hen forstår hva hen driver med? Er guiden fulgt, eller benyttes andre løsninger?
  For eksempel blir det det trekk her hvis det brukes static variabler som ikke er konstanter til bruk som kommunikasjon mellom ulike deler av programmet.

6c) Forbedringer

10 poeng.

6ci) Reset

2 poeng.

Oppgavesammendrag: Få reset-knappen til å flytte muldvarpen tilbake til midten ved klikk på reset-knappen.

```
public class WhacAMoleController {
    ...
    public void resetPressed(ActionEvent e) {
        model.reset();
    }
}

public class WhacAMoleView extends JPanel {
    ...
    public WhacAMoleView(WhacAMoleViewable model, WhacAMoleController controller) {
        ...
        this.add(new ButtonsPanel(controller), BorderLayout.SOUTH);
        ...
    }
}

public class ButtonsPanel extends JComponent {

    public ButtonsPanel(WhacAMoleController controller) {
        // Reset button
        JButton resetButton = new JButton("Reset");
        resetButton.addActionListener(controller::resetPressed);
        ...
    }
}
```

Sensorveiledning, reset. 1 poeng tildeles alltid dersom det fungerer, men 2 poeng gis bare dersom kallet går via kontrolleren som beskrevet.

- 1 poeng: resetPressed i kontroller korrekt implementert
- 1 poeng: ButtonsPanel tar inn kontroller, og kaller addActionListener

6cii) Poeng

3 poeng.

Oppgavesammendrag: vis poeng underveis i spillet.

```
public interface WhacAMoleViewable {
    ...
    Observable<Integer> getPoints();
}

public class WhacAMoleModel implements WhacAMoleViewable{
    ...
    private ControlledObservable<Integer> points = new
ControlledObservable<Integer>(0);
    ...

    public void reset() {
        ...
        points.setValue(0);
    }

    public void whac(Coordinate coordinate) {
        if (Objects.equals(MOLE, grid.get(coordinate))) {
            points.setValue(points.getValue() + 1);
            ...
        }
    }
    ...
    @Override
    public Observable<Integer> getPoints() {
        return points;
    }
}


public class WhacAMoleView extends JPanel {
    ...
    public WhacAMoleView(WhacAMoleViewable model, WhacAMoleController controller) {
        ...
        this.add(new HeadUpDisplay(model.getPoints()), BorderLayout.NORTH);
        ...
    }
}

public class HeadUpDisplay extends JComponent {

    private JLabel score;
    private Observable<Integer> points;
```

```
    public HeadUpDisplay(Observable<Integer> points) {
        ...
        // Score
        this.points = points;
        this.score = new JLabel();
        points.addObserver(this::updateScore);
        this.updateScore();
        ...
    }

    private void updateScore() {
        score.setText("" + points.getValue());
    }
}
```

Sensorveiledning (2 poeng tildeles automatisk dersom det fungerer i praksis, det tredje poenget krever at løsningen er i henhold til beskrivelsen)

- 1 poeng: poeng håndteres på en rimelig måte i modellen (observable ikke nødvendig)
- 1 poeng: poengene eksponeres kun som Observable via WhacAMoleViewable og tilhørende get-metode
- 1 poeng: visningen lytter til endringer i verdien/har lagt til en observatør som oppdatererer JLabel-objektet.

6ciii) Hover-effekt

1 poeng.

Oppgavesammendrag: La brukeren få visuell feedback når musen er over en klikkbar rute.

```
public class TileView extends JComponent implements MouseListener {
    ...
    private boolean mouseIsOver;
    ...

    @Override
    protected void paintComponent(Graphics g) {
        ...
        if (this.mouseIsOver && tile.getValue() == WhacAMoleModel.MOLE) {
            g.setColor(new Color(0xff, 0xff, 0xff, 0x60));
            g.fillRect(0, 0, getWidth(), getHeight());
        }
    }
    ...

    @Override
    public void mouseEntered(MouseEvent e) {
        this.mouseIsOver = true;
        this.repaint();
    }

    @Override
    public void mouseExited(MouseEvent e) {
        this.mouseIsOver = false;
        this.repaint();
    }
}
```

Sensorveiledning: Poenget tildeles dersom det fungerer i praksis. Vi godtar at det også har effekt på ruter uten muldvarp.

6civ) Timer

4 poeng.

Oppgavesammendrag: start en timer når første muldvarp klikkes. Vis nedtelling underveis, og gå til game over når tiden er ute.

```
public interface WhacAMoleViewable {
    ...
    Observable<String> getMessage();
}


public class WhacAMoleModel implements WhacAMoleViewable {
    ...
    private final static String READY_MSG = "Whac the mole!";
    private final static String GAME_OVER_MSG = "Game Over!";
    private final static int ROUND_TIME = 20;
    ...
    private GamePhase gamePhase = GamePhase.READY;
    private LocalDateTime gameEndTime = LocalDateTime.now();
    private ControlledObservable<String> message =
            new ControlledObservable<String>(READY_MSG);
    ...
    public void reset() {
        ...
        gamePhase = GamePhase.READY;
    }

    public void whac(Coordinate coordinate) {
        if (Objects.equals(MOLE, grid.get(coordinate))) {
            if (gamePhase == GamePhase.READY) {
                goToActivePhase();
            }
            ...
        }
    }

    public void onClockTick() {
        if (gamePhase == GamePhase.ACTIVE
                && LocalDateTime.now().isAfter(gameEndTime)) {
            goToGameOver();
        }
        updateMessage();
    }

    private void goToActivePhase() {
        gamePhase = GamePhase.ACTIVE;
        gameEndTime = LocalDateTime.now().plus(Duration.ofSeconds(ROUND_TIME));
    }
```

```
    private void goToGameOver() {
        gamePhase = GamePhase.GAMEOVER;
        grid.fill(BLANK);
    }

    private void updateMessage() {
        String messageString = switch (gamePhase) {
            case READY -> READY_MSG;
            case ACTIVE -> getTimeLeftToGameOverString();
            case GAMEOVER -> GAME_OVER_MSG;
        };
        message.setValue(messageString);
    }

    private String getTimeLeftToGameOverString() {
        Duration timeLeft = Duration.between(LocalDateTime.now(), gameEndTime);
        return String.format(
            "%02d.%03d",
            timeLeft.toSeconds(),
            timeLeft.toMillisPart()
        );
    }

    @Override
    public Observable<String> getMessage() {
        return this.message;
    }
}


public class WhacAMoleController implements ActionListener {

    private static final int MS_DELAY = 1000/60;
    ...
    public WhacAMoleController(WhacAMoleModel model) {
        ...
        Timer timer = new Timer(MS_DELAY, this);
        timer.start();
    }


    @Override
    public void actionPerformed(ActionEvent e) {
        model.onClockTick();
    }
}

public class HeadUpDisplay extends JComponent {
    ...
    private JLabel messageView;
    private Observable<String> message;
```

```
    public HeadUpDisplay(Observable<Integer> points, Observable<String> message) {
        // Message
        this.messageView = new JLabel();
        this.message = message;
        message.addObserver(this::updateMessage);
        updateMessage();
        ...
    }
    ...
    private void updateMessage() {
        messageView.setText(message.getValue());
    }
}
```

Sensorveiledning for timer.
Dersom alt fungerer, tildeles 4 poeng. Hvis ikke benyttes skjønn, men det skules til en slik fordeling (tilsvarer omtrent til grønne innrammede områdene). Dersom det ikke virker, blir det uansett ikke mer enn 2 poeng:

- 1 poeng: Kontrolleren oppretter en timer som periodisk kaller en metode i modellen
- 1 poeng: Modellen har en feltvariabel som bestemmer spillets tilstand, og denne modifiseres (når den skal) i whac-metoden, reset-metoden og klokkeslag-metoden.
- 1 poeng: En string som viser tiden som gjenstår produseres på en hensiktsmessig måte
- 1 poeng: Poengene vises

6d) Uavhengige oppgaver

6 poeng.

6di) AbstractGrid

3 poeng.

Oppgavesammendrag: Samle felles funksjonalitet for ControlledGrid og ListOfListsGrid i en abstrakt klasse, og la de konkrete klassene arve fra denne.

```
public abstract class AbstractGrid <E> implements GridWritable<E> {

    private int rows;
    private int cols;

    protected AbstractGrid(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
    }

    @Override public int getRows() { ... }
    @Override public int getCols() { ... }
    @Override public boolean coordinateIsOnGrid(...) { ... }
    @Override public Iterable<Coordinate> coordinates() { ... }
    @Override public Iterable<E> values() { ... }
    @Override public void fill(E value) { ... }
    @Override public void fill(Function<Coordinate, E> valueProducer) { ... }
    @Override public String toPrettyString(...) { ... }
    @Override public String toString() { ... }
}

public class ControlledGrid<E> extends AbstractGrid<E> implements GridObservable<E>
{
    ...
    public ControlledGrid(int rows, int cols, E defaultValue) {
        super(rows, cols);
        ...
    }

    @Override public void set(Coordinate coordinate, E value) { ... }
    @Override public E get(Coordinate coordinate) { ... }
    @Override public Observable<E> getObservable(...) { ... }
}

public class ListOfListsGrid <E> extends AbstractGrid<E> {
    ...
    public ListOfListsGrid(int rows, int cols, Function<Coordinate, E>
valueProducer) {
        super(rows, cols);
        ...
    }
```

```
    @Override public void set(Coordinate coordinate, E value) { ... }
    @Override public E get(Coordinate coordinate) { ... }
}
```

Sensorveiledning

- AbstractGrid (3 poeng)
    - 1 poeng: den abstrakte klassen er opprettet, kode er fjernet fra de konkrete klassene ListOfListGrid og ControlledGrid, og alt virker fremdeles.
    - 1 poeng: den abstrakte klassen er ansvarlig for feltvariablene for antall rader og antall kolonner, samt getRows og getCols -metodene. De konkrete klassene har dem ikke, men kaller super(rows, cols).
    - 1 poeng: den abstrakte klassen implementerer GridWritable (evt. IGrid), mens ListOfListGrid og ControlledGrid ikke gjør det (de bare utvider AbstractGrid)
- Dersom oppgaven er beskrevet løst med kommentarer uten at den faktisk er løst, gis det 1 poeng.

6dii) IteratorWrapper

3 poeng.

Dette er ment å være den vanskeligste oppgaven, og benytter konsepter vi bare har diskutert i forelesning uten å gjøre oppgaver om dem. Her handler det om å skaffe seg en intuisjon basert på hvordan koden blir brukt (basert på f. eks. bruken av den i grid'ene) og tolkning av hva koden gjør basert på de generiske typene og dokumentasjonen til Function.

Oppgavesammendrag: Skriv javadoc med eksempel for IteratorWrapper.

```
/**
 * Given an iterator which produce elements of type A and a function which maps
 * elements of type A to elements of type B, this creates an Iterator which
 * produce elements of type B. The iterator will consume the iterator given as
 * input, and apply the map function to elements when they are being consumed.
 *
 * <p>
 * For example, let {@code Iterator<String> sIt} be an iterator yielding String
 * objects, and say that you want to iterate of the length of the Strings rather
 * than their full String representations. Then you may create a WrapperIterator
 * that iterates over Integer objects instead:
 *
 * <p>
 * First step is to define a function which maps String to Integer. For example:
 *
 * <pre>
 * class MyClass {
 *     static Integer getLengthOfString(String s) {
 *         return s.length();
 *     }
```

```
 * }
 * </pre>
 *
 * <p>
 * Second step is to create the IteratorWrapper -object:
 *
 * <pre>{@code
 *
 * Iterator<Integer> it = new IteratorWrapper<String, Integer>(sIt,
MyClass::getLengthOfString);
 *
 * while (it.hasNext()) {
 *     int i = it.next();
 *     System.out.println(i);
 * }
 * }</pre>
 *
 * Both iterators {@code sIt} and {@code it} are exhausted when the code finish.
 *
 * @param base the source iterator
 * @param map  function which transforms elements of type A into elements of
 *             type B
 */
```

Sensorveiledning

- Dokumentasjon (3 poeng)
    - 1 poeng: eksempel som er enten selvforklarende eller blir forklart (det gir ingen poeng å benytte eksempelet fra grid-klassene hvis det ikke blir forklart)
    - 2 poeng: Skjønn. Demonstrerer kandidaten at hen forstår hva spørsmålet dreier seg om? Det skal ikke trekkes for feil som er primært språklige (som skrivefeil eller grammatikk) men fokus skal være på hvorvidt kandidaten velger gode uttrykk og begreper fra fagfeltet for å beskrive det hen mener. Kommentarene kan være skrevet enten for klassen eller for konstruktøren eller for begge.