

Grundlæggende programmering

10 sider i alt

4 timers skriftlig prøve fredag den 13. juni 2003 kl. 9:00 – 13:00

Alle skriftlige hjælpemidler er tilladte til denne eksamen, men ikke elektroniske.

Besvarelsen bedømmes efter 13-skalaen, vægten af de enkelte opgaver og spørgsmål er angivet i parentes i overskriften.

Det tilrådes, at du læser hele opgavesættet igennem inden du begynder løsningen af det.

1for-sætningen (5%)	2
1.1Hvad er uddata? (5%)	2
2Fra do-while til while (5%)	2
2.1Omskriv løkken (5%).....	2
3Metoder og statiske felter (15%)	3
3.1Skriv konstruktøren (5%).....	3
3.2Skriv rapporteringsmetoderne (10%).....	4
4Referencer (5%)	4
4.1Hvad er uddata? (5%).....	4
5Exceptions (15%)	4
5.1Hvad er uddata? (5%).....	5
5.2Skriv koden til klassen MyFile (10%).....	5
6Hændelser og GUI (10%)	6
6.1Gør knapperne interaktive (10%).....	6
7Arv i fragtvirkomheden (20%)	7
7.1Vis klassehierarkiet (10%).....	7
7.2Skriv klassen Køretøj (5%).....	8
7.3Skriv klassen Lastvogn (5%).....	8
8Gruppeadministrationssystemet (25%)	8
8.1Skriv klassen Student (10%).....	9
8.2Skriv klassen Group (15 %).....	9

I opgavesættet er det brugt følgende notation og formulering:

ÿ "?" repræsenterer kode, der ikke er vist. Du kan bruge samme notation i din bevarelse

ÿ "udskrivning på konsollen" betyder udskrift ved hjælp af `print`- og `println`-metoderne fra `System.out`-objektet

Kommentarer i ramme er løsningsforslag – der kan være andre løsninger, som er (mindst) lige så gode.

1 for-sætningen (5%)

En metode `print` ser således ud:

```
public void print(int m) {
    for (int i = 0; i < m; i++) {
        char x;
        if (i % 2 == 0) x =    ; else x = + ;
        for (int j = 0; j < m; j++) System.out.print(x);
        System.out.println();
    }
}
```

1.1 Hvad er uddata? (5%)

Hvad udskriver metoden på konsollen, hvis den kaldes med `print(4)`;

```
++++
++++
```

2 Fra do-while til while (5%)

En metode `foo` ser således ud:

```
public static int foo(int n) {
    int max = 0;
    if (n > 0) {
        do {
            n ;
            max = max + n;
        } while (n > 0);
    }
    return max;
}
```

2.1 Omskriv løkken (5%)

Omskriv metoden `foo`, så den benytter en `while`-løkke i stedet for en `do-while`-løkke.

```
public int foox(int n) {
    int max = 0;
    while (n > 0) {
        n ;
        max = max + n;
    }
    return max;
}
```

3 Metoder og statiske felter (15%)

En klasse `Måling` er erklæret på følgende måde

```
class Måling {  
    public final static double MAX = 100.0; //Den højest mulige  
    værdi  
    public final static double MIN = 0.0; //Den lavest mulige  
    værdi  
  
    private static int antal = 0; // Antallet af målinger hidtil  
    private static double maximum = MIN; // Den hidtil højeste værdi  
    private static double minimum = MAX; // Den hidtil laveste værdi  
  
    private double data;  
    private int dataNr;  
  
    public Måling(double d) {  
        ?  
    }  
  
    public static void report() {  
        ?  
    }  
  
    public String toString() {  
        ?  
    }  
}
```

Denne klasse kan prøves ved fx følgende metode:

```
public static void staticTest(int size) {  
    Måling[] m = new Måling[size];  
    for (int i = 0; i < size; i++) {  
        m[i] = new Måling(Math.random() * Måling.MAX);  
    }  
  
    Måling.report();  
    for (int i = 0; i < size; i++) {  
        System.out.println(m[i].toString());  
    }  
}
```

hvor kaldet `staticTest(5)` fx kan give følgende resultat (visse decimaler er skåret fra)

```
Antal = 5, Højeste værdi = 95.78339382, Laveste værdi = 10.96159341  
Data nr 0: 91.67579071  
Data nr 1: 95.78339382  
Data nr 2: 10.96159341  
Data nr 3: 51.02456100  
Data nr 4: 89.13057601
```

3.1 Skriv konstruktøren (5%)

Skriv konstruktøren til klassen `Måling`

3.2 Skriv rapporteringsmetoderne (10%)

Skriv de to metoder `report` og `toString`, således at `report` udskriver den første af de viste linjer i udskriften, mens hvert kald af `toString` returnerer en af de 5 linjer med data.

```
public Måling(double d) {
    data = d;
    dataNr = antal;
    antal++;
    if (data > maximum) maximum = data;
    if (data < minimum) minimum = data;
}

public static void report() {
    System.out.println(
        "Antal = " + antal + ", " +
        "Højeste værdi = " + maximum + ", " +
        "Laveste værdi = " + minimum);
}

public String toString() {
    return "Data nr " + dataNr + ": " + data;
}
```

4 Referencer (5%)

Betragt koden nedenfor:

```
public static void refs() {
    String s1 = "Abba";
    String s2 = "Beatles";
    String s3 = "C21";
    String s4 = "Doors";

    System.out.println("A: " + s1 + "/" + s2 + "/" + s3 + "/" + s4);
    s2 = s1;
    String sx = s2;
    s2 = s3;
    s3 = s4;
    System.out.println("B: " + s1 + "/" + s2 + "/" + s3 + "/" + s4);
}
```

4.1 Hvad er uddata? (5%)

Hvad udskriver metoden på konsollen, hvis den kaldes med `refs()` ?

```
A: Abba/Beatles/C21/Doors
B: Abba/C21/Doors/Doors
```

5 Exceptions (15%)

Antag, at der findes en klasse `MyFile`, der er defineret således:

```
public class MyFile {
    ?

    public boolean open(String fileName) {
```

```

    ?
}

public String nextLine() {
    ?
}

}

```

Metoden `open` forsøger at åbne tekst-filen med navnet `fileName`. Hvis dette lykkes, returnerer kaldet værdien `true`, ellers returnerer det værdien `false`. Metoden `nextLine` returnerer filens næste line, hvis en sådan ikke findes returneres værdien `null`.

5.1 Hvad er uddata? (5%)

Antag, at en stump kode ser således ud.

```

MyFile file = new MyFile();

if (!file.open("test.txt")) {
    throw new Error("Filen kunne ikke åbnes");
}

String s = file.nextLine();
while (s != null) {
    System.out.println(s + "\n");
    s = file.nextLine();
}

```

Hvad vil uddata blive,

Ø dersom filen `test.txt` ikke findes?

```

Exception in thread "main" java.lang.Error: Filen kunne ikke ?bnes
    at Opg5.doIO(Opg5.java:14)
    at Eksamensstart.main(Eksamensstart.java:13)
[JCC-note: det forventes ikke, at de studerende kan gengive stakken korrekt]

```

Ø dersom filen `test.txt` indeholder følgende tekst

Her er en
tekst med

fire linier

```

Her er en (1)
tekst med (2)
(3)
fire linier (4)

```

5.2 Skriv koden til klassen `MyFile` (10%)

Skriv koden til klassen `MyFile`, så metoderne kommer til at virke som ovenfor beskrevet.

```

class MyFile {

    private FileReader fileReader;
    private BufferedReader bufferReader;
    private String line;

    public boolean open(String fileName) {
        try {
            fileReader = new FileReader(fileName);
            bufferReader = new BufferedReader(fileReader);

```

```

        return true;
    } catch (FileNotFoundException e) {
        return false;
    }
}

public String nextLine() {
    try {
        return bufferedReader.readLine();
    } catch (IOException e) {
        return null;
    }
}
}
}

```

6 Hændelser og GUI (10%)

Klassen `MyFrame` er defineret således:

```

class MyFrame extends JFrame {

    private JTextField textField = new JTextField(20);
    private JButton storButton = new JButton("Store");
    private JButton småButton = new JButton("Små");

    public MyFrame() {
        Container pane = getContentPane();
        pane.add(textField, BorderLayout.NORTH);
        JPanel buttons = new JPanel(new GridLayout());
        buttons.add(storButton);
        buttons.add(småButton);
        pane.add(buttons, BorderLayout.SOUTH);

        setSize(200, 75);
        show();
    }
}

```



Ved oprettelsen af et objekt af denne klasse vil der på brugerens skærm komme et vindue til syne, mægt til det der er vist her på siden. Men ved klik på knapperne vil der ikke ske noget.

6.1 Gør knapperne interaktive (10%)

Tilføj til klassen `MyFrame` den kode der gør, at

- et klik på knappen `storButton` får teksten i feltet `textField` til at blive vist med litter store bogstaver
- et klik på knappen `småButton` får teksten i feltet `textField` til at blive vist med litter små bogstaver

```

public class MyFrame extends JFrame {

    private JTextField textField = new JTextField(20);
    private JButton storButton = new JButton("Store");
    private JButton småButton = new JButton("Små");

```

```
public MyFrame() {
    Container pane = getContentPane();
    pane.add(textField, BorderLayout.NORTH);
    JPanel buttons = new JPanel(new GridLayout());
    buttons.add(storButton);
    buttons.add(småButton);
    pane.add(buttons, BorderLayout.SOUTH);

    storButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            textField.setText(textField.getText().toUpperCase());
        }
    });

    småButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            textField.setText(textField.getText().toLowerCase());
        }
    });

    setSize(200, 75);
    show();
}
}
```

7 Arv i fragtvirksomheden (20%)

En landsdækkende transportvirksomhed ejer en større bilpark med forskellige typer af køretøjer. Virksomhedens programmør er ved at udvikle et system til registrering af disse køretøjer, hvoraf der findes følgende kategorier:

- køretøjer til godsfragt (lastvogne), som er kendetegnet ved en maksimal nyttelast
- køretøjer til personfragt (personkøretøjer), som er kendetegnet ved det antal personer, som køretøjet kan medbringe. Der findes to varianter af disse køretøjer, nemlig
 - personbiler, der har en fast tilknyttet chauffør
 - busser, der ikke har nogen fast tilknyttet chauffør

Uanset kategori er alle køretøjer kendetegnet ved et fabrikat, et købsår og et registreringsnummer,

Programmøren vil i sit Java-program lade hver af disse typer af køretøjer være repræsenteret af sin egen klasse, hvor disse klasser arrangeres i et klassehierarki.

7.1 Vis klassehierarkiet (10%)

Vis klassehierarkiet og felterne for klasserne

- Køretøj
- Personkøretøj
- Personbil
- Bus
- Lastvogn

Klassehierarkiet kan formuleres som Java-kode eller som UML-diagram efter eget valg. Du skal kun medtage de felter, som følger af ovenstående beskrivelse.

7.2 Skriv klassen *Køretøj* (5%)

Skriv klassen *Køretøj*. Der skal kun medtages felter, konstruktør og metoden `toString`. Et kald til denne metode skal returnere en tegnstreng bygget op på følgende måde:

```
Fabrikat=Ford, år=1998, rg.nr=TX 98546
```

7.3 Skriv klassen *Lastvogn* (5%)

Skriv klassen *Lastvogn*. Der skal kun medtages felter, konstruktør og metoden `toString`. Et kald til denne metode skal returnere en tegnstreng bygget op på følgende måde:

```
Fabrikat=Ford, år=1998, rg.nr=TX 98546, maxLast=8000 kg
```

```
class Koeretoej {  
  
    private String fabrikat;  
    private int købsår;  
    private String registreringsnr;  
  
    public Koeretoej (String fab, int år, String regnr) {  
        fabrikat = fab;  
        købsår = år;  
        registreringsnr = regnr;  
    }  
  
    public String toString() {  
        return "Fabrikat=" + fabrikat + ", år=" + købsår + ", rg.nr=" + registreringsnr;  
    }  
}  
  
class Lastvogn extends Koeretoej {  
  
    private int maxLast;  
  
    public Lastvogn(String fab, int år, String regnr, int max) {  
        super(fab, år, regnr);  
        maxLast = max;  
    }  
  
    public String toString() {  
        return super.toString() + ", maxLast=" + maxLast + " kg";  
    }  
}
```

8 Gruppeadministrationssystemet (25%)

Et system til administration af projektgrupper på et universitet har blandt andet følgende klasser:

```
class Student {  
  
    private String firstName;  
    private String lastName;  
    private Group group;  
  
    public Student(?) {  
        ?  
    }  
}
```

```

    public void setGroup(?) {
        ?
    }
}

class Group {
    ?

    public void addStudent(Student student) {
        ?
    }

    public void printGroup(?) {
        ?
    }
}

```

8.1 Skriv klassen *Student* (10%)

Skriv til klassen `Student` parameterlisten og kroppen til

- konstruktøren, sådan at alle felter bliver initialiseret (dette indebærer, at den studerende skal have både fornavn og efternavn, men ikke at det er tilknyttet nogen gruppe)
- metoden `setGroup`, sådan at den studerende bliver tilknyttet en gruppe. Relationen mellem studerende og gruppe skal være tovejs, så metoden skal kalde gruppens `addStudent`-metode. Du må gerne antage, dels at medlemmet ikke i forvejen er med i nogen gruppe, dels at `setGroup` ikke bliver kaldt med parameteren `null`.

8.2 Skriv klassen *Group* (15 %)

Skriv klassen `Group`, herunder

- erklæringerne til felterne, sådan at referencerne til gruppens studerende indeholdes i et array (der er højst 18 studerende på et hold)
- gruppens nummer, et entydigt id, som tildeles gruppen ved oprettelsen
- konstruktøren, hvis den er nødvendig
- metoden `addStudent`, som tilføjer en studerende til gruppen
- metoden `printGroup`, som skriver en liste over gruppens studerende på konsollen

```

public class Student {

    private String firstName;
    private String lastName;
    private Group group;

    public Student(String f, String l) {
        firstName = f;
        lastName = l;
        group = null;
    }

    public void setGroup(Group g) {
        //pre: (group == null) && (g != null)
        group = g;
        group.addStudent(this);
    }
}

```

```
public String toString() {
    return firstName + " " + lastName;
}
}

public class Group {

    private static int MAX_STUDENTS = 18;
    private static int noOfGroups = 0;

    private int no;
    private Student[] all = new Student[MAX_STUDENTS];
    private int used = 0;

    public Group() {
        no = ++noOfGroups;
    }

    public void addStudent(Student student) {
        if (used == MAX_STUDENTS) throw new Error("Too many students in group");
        all[used] = student;
        used++;
    }

    public void printGroup() {
        System.out.println("Group no " + no);
        for (int i = 0; i < used; i++) {
            System.out.println(" " + all[i].toString());
        }
    }
}
}
```