NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4100 Objektorientert programmering

Faglig kontakt under eksamen: Hallvard Trætteberg

Tlf.: 91897263

Eksamensdato: 6. juni

Eksamenstid (fra-til): 9.00-13.00

Hjelpemiddelkode/Tillatte hjelpemidler: C

Kun ”Big Java”, av Cay S. Horstmann, er tillatt.

Annen informasjon:

Oppgaven er utarbeidet av faglærer Hallvard Trætteberg og kvalitetssikret av Ragnhild Kobro Runde (Ifi, UiO).

Målform/språk: Bokmål

Antall sider: 5

Antall sider vedlegg: 1

Kontrollert av:

_____

Dato Sign

Dersom du mener at opplysninger mangler i en oppgaveformulering, gjør kort rede for de antagelser og forutsetninger som du finner nødvendig. Hvis du i en del er bedt om å *implementere* klasser og metoder og du ikke klarer det (helt eller delvis), så kan du likevel *bruke* dem i senere deler.

Del 1 – Teori (20%)

Gitt følgende klasse:

```
public class MysticalObject {

	private int number;

	public MysticalObject(int number) {
		this.number = number;
	}

	public void step() {
		number = -(number - (int) Math.signum(number));
	}

	public int getNumber() {
		return number;
	}
}
```

Se vedlegget for dokumentasjon til **Math.signum**-metoden.

a) Tegn et objekttilstandsdiagram for et **MysticalObject**-objekt som er instansiert med **new MysticalObject(3)**, som viser oppførselen til **getNumber**- og **step** methods.

b) Objekttilstandsdiagrammer kan brukes til å beskrive oppførselen til et objekt. Angi en viktig fordel og en ulempe/begrensning ved slik bruk.

c) Java krever at en deklarerer *typen* til alle felt, variabler og parametre (i motsetning f.eks. Python, Javascript og Matlab). Hva er de viktigste fordelene dette gir?

d) I koden over så står det **(int)** foran kallet til **Math.signum**. Hva kalles dette (denne typen *uttrykk*) og hvordan virker det (påvirker det utførelsen) i dette tilfellet? Den samme syntaksen kan også brukes i tilfeller hvor typen er en klasse eller et grensesnitt. Hvordan virker det da?

Del 2 – Klasser (30%)

I denne oppgaven skal du implementere klasser og metoder for en quiz, altså sekvenser med spørsmål og svar. Et eksempel på spørsmål og svar-sekvens er vist under. Som eksemplet viser, så kan det kan være ulike typer svar og en kan ha svar-alternativer.

```
Hva heter hovedstaden i Norge?
Oslo
Hva slags ost er månen laget av? 1. Camembert 2. Roquefort 3. Brie
2
```

```
Hvor høy er Galdhøpiggen? 1. 2469 2. 2471
2
Feil, prøv igjen!
2469
Er Java gøy?
ja
```

Du skal implementere én klasse for spørsmål og tilhørende svar kalt **Question** og én klasse for quiz-en kalt **Quiz**. Oppgaven videreføres i resten av oppgavesettet, så det kan være greit å lese gjennom alle delene før du begynner å jobbe med din løsning.

a) Du skal først implementere **Question**-klassen. Klassen skal støtte spørsmål, svar og eventuelle svar-alternativer, som alle er tekster. Definer felt for disse og skriv en (eller flere) konstruktør(er) som initialiserer dem. Merk at et spørsmål *må* ha et svar, men *trenger ikke* ha svar-alternativer. Utløs unntak for tilfeller som du synes er relevante.

b) Lag følgende metoder i **Question**-klassen for å stille spørsmål og sjekke svar:

- **askQuestion(PrintStream)**: stiller spørsmålet ved å skrive tekst til **PrintStream**-argumentet. Eventuelle svar-alternativer skal også skrives ut. Nummerer alternativene fra 1 og utover, så det er lettere å skrive inn nummeret som svar.
- **checkAnswer(String answer)**: sjekker svaret angitt som argument og returnerer **true** om det er riktig, og **false** ellers. Dersom spørsmålet har svar-alternativer, så skal det både være lov å angi svaret eller *nummeret på svar-alternativet* (fortsatt som en **String**). Ta høyde for tilfellet hvor et spørsmål har et tall som riktig svar og har svar-alternativer, slik som i Galdhøpiggen-spørsmålet i eksemplet over.

c) Det er ofte et skjønnspørsmål om en trenger get- og set methods og evt. andre metoder for å lese og endre tilstanden i et objekt. Begrunn hvilke slike metoder som trengs (om noen) i **Question**-klassen og implementer dem.

d) **Quiz**-klassen representerer en sekvens med spørsmål. Skriv kode for klassen med metoder for å:

- legge til **Question**-objekter
- stille spørsmål med utskrift til skjerm (**System.out**) og lese inn svar fra tastaturet (**System.in**) helt til det er svart riktig på hvert spørsmål, omtrent som vist i eksemplet i innledningen.

e) Skriv nødvendige metoder for å kunne *kjøre* **Quiz**-klassen (med noen eksempelspørsmål) som et hovedprogram iht. reglene som Java har for dette. Når man kjører **Quiz**-klassen, skal altså brukeren bli stilt spørsmålene og få anledning til å svare, omtrent som vist i eksemplet i innledningen.

Del 3 – Input/output og unntak (IO) (10%)

Utvid **Quiz**-klassen med en metode for å initialisere **Quiz**-objektet med spørsmål (og tilhørende svar og evt. svar-alternativ) fra fil. Følgende format skal støttes:

- Spørsmål, svar og evt. svar-alternativer har én linje hver.
- Spørsmålet kommer først, så svaret og deretter evt. svar-alternativer.
- Før neste spørsmål er det en tom linje.

Eksempelfil tilvarende eksemplet i innledningen, med kommentarer (som ikke er en del av filinnholdet) i høyre kolonne:

```
Hva heter hovedstaden i Norge?
Oslo

Hva slags ost er månen laget av?
Roquefort
Camembert
Roquefort
Brie

Hvor høy er Galdhøpiggen?
2469
2469
2471

Er Java gøy?
ja
```

1. spørsmål
riktig svar
skillelinje
2. spørsmål
riktig svar
1. svar-alternativ
2. svar-alternativ
3. svar-alternativ
skillelinje
3. spørsmål
riktig svar
1. svar-alternativ
2. svar-alternativ
skillelinje
4. spørsmål
riktig svar

Del 4 – Arv (15%)

I denne delen skal du bruke *arv* for å håndtere ulike typer svar på en ryddigere måte. Det skal være én klasse for frie tekst-svar kalt **StringQuestion**, en klasse for tekst-svar med svar-alternativer kalt **StringOptionsQuestion**, og en klasse for ja/nei-spørsmål kalt **BooleanQuestion**. Det som er felles for disse klassene, f.eks. håndtering av selve spørsmålsteksten, skal samles i **Question**-superklassen. Bortsett fra at **Question**-klassen ikke skal kunne instansieres, så skal *bruken* av den være som i del 3, inkludert de to **askQuestion**- og **checkAnswer** methods fra del 2. Prøv å strukturere klassene dine, så det blir *minst mulig duplisert kode*. Du står selvsagt fritt til å definere andre metoder som trengs i løsningen din.

En del av koden vil være lik tidligere kode. Du kan selv velge om du vil skrive den på nytt, eller beskrive presist hvordan tidligere skrevet kode(tekst) kopieres inn i de nye klassene.

a) Implementer først **Question**-superklassen og de tre klassene **StringQuestion**, **StringOptionsQuestion** og **BooleanQuestion** kun med konstruktører.

b) Implementer de to methods **askQuestion** og **checkAnswer**, slik at alle **Question**-objekter (egentlig instanser av en av de tre andre klassene) i praksis virker som i del 2.

c) Reimplementer metoden i **Quiz** for innlesing av spørsmål fra fil, slik at den virker med de nye **Question**-subklassene.

Del 5 – Trinnvis utførelse (15%)

I denne delen skal **Quiz**-klassen endres slik at den holder rede på tilstanden til en runde med spørsmål og svar, men lar en *annen* (hovedprogram)klasse styre fremdriften. Følgende tre metoder skal brukes til å styre fremdriften:

- **start(boolean mode, PrintStream out, InputStream in)**: starter quiz-en (men ingen spørsmål stilles ennå). **mode**-argumentet angir om et galt besvart spørsmål gjentas *med en gang* (**mode**=**false**), eller *etter at* alle etterfølgende spørsmål er stilt (**mode**=**true**). **out**-

argumentet er strømmen som spørsmål skal skrives til (gjøres i **doQuestion**). **in**-argumentet er strømmen som svarene skal leses fra (gjøres også i **doQuestion**).

- **doQuestion()**: Stiller ett (neste) spørsmål og leser ett svar(forsøk). Hvilket spørsmål som stilles avgjøres av hvilke som ennå ikke er stilt og besvart riktig og **mode**-verdien som ble gitt til **start**-metoden. Metoden returnerer antall spørsmål som *ennå ikke er riktig besvart*.
- **stop()**: stopper quiz-en og returnerer hvor mange spørsmål som *ble riktig besvart*.

Følgende kode eksemplifiserer hvordan (denne versjonen av) **Quiz**-klassen er ment å bli brukt:

```
Quiz quiz = new Quiz();
// initialiser fra fil her (ikke vist)
quiz.start(true, System.out, System.in); // start quiz
while (quiz.doQuestion() > 0); // still spørsmål så lenge flere gjenstår
quiz.stop();
```

Implementer **start**-, **doQuestion**- og **stop** methods.

Del 6 – Grensesnitt (10%)

Det er ønskelig å kunne støtte ulike filformat for quiz-er. Forklar med tekst og kode hvordan du vil bruke Java-grensesnitt for å gjøre det enkelt å bytte mellom ulike format og hvordan filinnlesingskoden du allerede har skrevet kan utgjøre (implementasjonen av) standardformatet. Merk at du ikke skal implementere nye format, kun vise hvordan grensesnitt-mekanismen gjør det enkelt.

NTNU – Trondheim
Norwegian University of
Science and Technology

Department of computer and information science

Examination paper for TDT4100
Object-oriented programming with Java

Academic contact during examination: Hallvard Trætteberg

Phone: 91897263

Examination date: 6. June

Examination time (from-to): 9:00-13:00

Permitted examination support material: C

Only ”Big Java”, by Cay S. Horstmann, is allowed.

Other information:

This examination paper is written by teacher Hallvard Trætteberg, with quality assurance by Ragnhild Kobro Runde (Ifi, UiO).

Language: English

Number of pages: 5

Number of pages enclosed: 1

Checked by:

Date Signature

If you feel necessary information is missing, state the assumptions you find it necessary to make. If you are not able to *implement* classes and method that a part asks for, you may still use these classes and methods later.

Part 1 – Theory (20%)

Given the following class:

```
public class MysticalObject {

	private int number;

	public MysticalObject(int number) {
		this.number = number;
	}

	public void step() {
		number = -(number - (int) Math.signum(number));
	}

	public int getNumber() {
		return number;
	}
}
```

See the appendix for documentation for the **Math.signum** method.

a) Draw an object state diagram for a **MysticalObject** objekt that is instantiated with **new MysticalObject(3)**, that shows the behaviour of the **getNumber** and **step** methods.

b) Object state diagrams can be used for describing the behaviour of an object. Provide one advantage and one disadvantage for this kind of usage.

c) Java requires that you declare the *type* of all fields, variables and parameters (as opposed to e.g. Python, Javascript and Matlab). What are the main advantages of having this requirement?

d) In the code above, the term **(int)** precedes the call to **Math.signum**. What is this construct called (this kind of *expression*) and how does it work (affect the evaluation) in this particular case? The same syntax can also be used in cases where the type is a class or interface. How does is work in that case?

Part 2 – Classes (30%)

In this part you will implement classes and methods for a quiz, i.e. sequences of questions and answers. An example of such a question and answer sequence is shown below. As shown, there can be different kinds and answers and there may be options to select from.

```
What is the capital of Norway?
Oslo
What kind of cheese is the moon made of? 1. Camembert 2. Roquefort 3. Brie
```

```
2
How high is Galdhøpiggen? 1. 2469 2. 2471
2
Wrong, try again!
2469
Is Java fun?
yes
```

You will implement one class for questions and its answer named **Question** and one class for the quiz itself named **Quiz**. These classes are developed further in the rest of this exam, so it may be a good idea to read through all parts before starting work on your solution.

a) You will first implement the **Question** class. This class must support a question, answer and answer options, all of which are text. Define fields for these and write one (or more)constructor(s) that initialises them. Note that a question *must* have an answer, but *does not need* answer options. Use exceptions for cases you find relevant.

b) Write the following methods in the **Question** class for asking questions and checking answers:

- **askQuestion(PrintStream)**: asks the question by writing text to the **PrintStream** argument. Answer options must also be written, if there are any. Number the options from 1 and onward, so it is easier to provide the number as the answer.
- **checkAnswer(String answer)**: checks that the answer provided as the argument and returns **true** if it is correct, and **false** otherwise. If the question has options, then both the answer and the *option number* should be considered (still as a **String**). Take care to handle the case where the correct answer is a number and there also are options, like the Galdhøpiggen question in the example above.

c) You usually need to consider if there is a need for get and set methods or other methods for reading and change object state. Argue which such methods are needed (if any) in the **Question** class and implement them.

d) The **Quiz** class represents a sequence of questions. Write code for this class with methods for:

- adding **Question** objects
- asking questions by writing to and reading from the console (**System.out** and **System.in** respectively) until each question has been answered correctly, like what is shown in the example in the introduction

e) Write necessary methods for *running* the **Quiz** class (with sample questions) as a program (application) according to Java's rules for this. When the **Quiz** class is run, the user should be asked the questions and be able to answer, like shown in the example in the introduction.

Part 3 – Input/output and exceptions (IO) (10%)

Extend the **Quiz** class with a method for initialising the **Quiz** object with questions (and their answers and options) from a file. The following format must be supported:

- The question, answer and options have one line each.
- The question comes first, followed by the answer and options (if any).
- Before the following question there is an empty line.

File contents corresponding to the example in the introduction, with comments (not part of the file) in the right column:

```
What is the capital of Norway?
Oslo

What kind of cheese is the moon made
of?
Roquefort
Camembert
Roquefort
Brie

How high is Galdhøpiggen?
2469
2469
2471

Is Java fun?
yes
```

1. question
correct answer
empty line
2. question
correct answer
1. answer option
2. answer option
3. answer option
empty line
3. question
correct answer
1. answer option
2. answer option
empty line
4. question
correct answer

Part 4 – Inheritance (15%)

In this part you will use inheritance for handling different kinds of answers in a more structured way. There must be one class for free text answers named **StringQuestion**, one class for tekst answers with options named **StringOptionsQuestion**, and one class for yes/no questions named **BooleanQuestion**. The common behaviour for these classes, e.g. handling of the question itself, must be in the **Question** superclass. Besides the fact that it should not be possible to instantiate the **Question** class, its usage should be the same as for Part 3, including the two **askQuestion** and **checkAnswer** methods from Part 2. Try to structure your classes so there is as *little code duplication as possible*. Feel free to define other methods that you find useful in your solution.

Some of the code will be the same as written in previous parts. You can choose to write it again or indicate precisely how it should be copied into the new classes.

a) First, implement the **Question** superclass and the three classes **StringQuestion**, **StringOptionsQuestion** and **BooleanQuestion** with constructors only.

b) Implement the two methods **askQuestion** and **checkAnswer**, so all **Question** objects (instances of one of the other three classes) behave like in Part 2.

c) Reimplement the method in **Quiz** for reading questions from fil, so it works with the new **Question** subclasses.

Part 5 – Stepwise execution (15%)

In this part, the **Quiz** class must be modified to keep track of the state of a round of questions and answers, but lets *another* (application) class control the progress. The following three methods are used to control the progress:

- **start(boolean mode, PrintStream out, InputStream in)**: starts the quiz (but does not ask any questions yet). The **mode** argument determines if an incorrectly answered question is repeated *at once* (**mode=false**), or *after* all subsequent questions are asked (**mode=true**). The **out** argument is the stream the questions are written to (by **doQuestion**). The **in** argument is the stream the answers are read from (by **doQuestion**).
- **doQuestion()**: Asks one (the next) question and reads one answer (attempt). Which question that is asked is determined by which ones haven't yet been asked and correctly answered and the **mode** value provided to the **start** method. The method must return the number of questions that *haven't yet been answered correctly*.
- **stop()**: stops the quiz and returns how many questions that *were answered correctly*.

The following code exemplifies how (this version of) the **Quiz** class is meant to be used:

```
      Quiz quiz = new Quiz();
      // initialise from file (not shown)
      quiz.start(true, System.out, System.in); // start quiz
      while (quiz.doQuestion() > 0); // ask questions as long as unanswered ones
remain
      quiz.stop();
```

Implement the **start**, **doQuestion** and **stop** methods.

Part 6 – Interface (10%)

You wish to support several file formats for quizzes. Explain with text and code how you would use Java interfaces for making it easy to switch between different formats and how the existing code for reading from file can be used as the (implementation of the) default format. Note that you do not need to implement other formats, just show how the interface mechanism makes it easy to do.

NTNU
Kunnskap for ei betre verd

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgåve i TDT4100
Objektorientert programmering

Fagleg kontakt under eksamen: Hallvard Trætteberg

Tlf.: 91897263

Eksamensdato: 6. juni

Eksamenstid (frå-til): 9.00-13.00

Hjelpemiddelkode/Tillatne hjelpemiddel: C

Berre ”Big Java”, av Cay S. Horstmann, er tillaten.

Annan informasjon:

Oppgåva er utarbeidd av faglærer Hallvard Trætteberg og kvalitetssikra av Ragnhild Kobro Runde (Ifi, UiO).

Målform/språk: Nynorsk

Sidetal: 5

Sidetal vedlegg: 1

Kontrollert av:

_____

Dato Sign

Om du meiner at opplysningar manglar i ein oppgåveformulering, gjer kort greie for dei antakingar og føresetnader som du finn naudsynt. Om du i ein del er beden om å *implementere* klasser og metodar og du ikkje klarar det (heilt eller delvis), så kan du likevel *nytte* dei i seinare delar.

Del 1 – Teori (20%)

Gitt fylgjande klasse:

```
public class MysticalObject {

	private int number;

	public MysticalObject(int number) {
		this.number = number;
	}

	public void step() {
		number = -(number - (int) Math.signum(number));
	}

	public int getNumber() {
		return number;
	}
}
```

Sjå vedlegget for dokumentasjon til **Math.signum**-metoden.

a) Teikn eit objekttilstandsdiagram for eit **MysticalObject**-objekt som er instansiert med **new MysticalObject(3)**, som viser oppførselen til **getNumber**- og **step**-metodane.

b) Objekttilstandsdiagram kan nyttast til å beskrive oppførselen til eit objekt. Angi ein viktig fordel og ei ulempe ved slik bruk.

c) Java krev at ein deklarerer *typen* til alle felt, variablar og parameter (i motsetnad t.d. Python, Javascript og Matlab). Kva er dei viktigaste fordelane dette gir?

d) I koden over så står det **(int)** føre kallet til **Math.signum**. Kva kallast dette (denne typen *uttrykk*) og korleis verkar det (påverkar det utføringa) i dette tilfellet? Den same syntaksen kan og nyttast i tilfelle kor typen er ein klasse eller eit grensesnitt. Korleis verkar det da?

Del 2 – Klasser (30%)

I denne oppgåva skal du implementere klasser og metodar for ein quiz, altså sekvensar med spørsmål og svar. Eit døme på spørsmål og svar-sekvens er vist under. Som dømet viser, så kan det kan vere ulike typar svar og ein kan ha svar-alternativ.

```
Kva heiter hovudstaden i Noreg?
Oslo
Kva slags ost er månen laga av? 1. Camembert 2. Roquefort 3. Brie
2
```

```
Kor høg er Galdhøpiggen? 1. 2469 2. 2471
2
Feil, prøv igjen!
2469
Er Java gøy?
ja
```

Du skal implementere ein klasse for spørsmål og tilhøyrande svar kalla **Question** og ein klasse for quiz-en kalla **Quiz**. Oppgåva førast vidare i resten av oppgåvesettet, så det kan vere greitt å lese gjennom alle delane før du tar til å jobbe med di løysing.

a) Du skal fyrst implementere **Question**-klassen. Klassen skal støtte spørsmål, svar og eventuelle svar-alternativ, som alle er tekstar. Definer felt for desse og skriv ein (eller fleire) konstruktør(ar) som initialiserer dei. Merk at et spørsmål *må* ha eit svar, men *treng ikkje* ha svar-alternativ. Utløys unntak for tilfelle som du synst er relevante.

b) Lag fylgjende metodar i **Question**-klassen for å stille spørsmål og sjekke svar:
- **askQuestion(PrintStream)**: stiller spørsmålet ved å skrive tekst til **PrintStream**-argumentet. Eventuelle svar-alternativ skal og skrivast ut. Nummerer alternativa frå 1 og utover, så det er lettare å skrive inn nummeret som svar.
- **checkAnswer(String answer)**: sjekkar svaret gitt som argument og returnerer **true** om det er riktig, og **false** elles. Dersom spørsmålet har svar-alternativ, så skal det både vere lov å skrive svaret og angi *nummeret på svar-alternativet* (enno som ein **String**). Ta høgde for tilfellet kor eit spørsmål har eit tal som riktig svar og har svar-alternativ, slik som i Galdhøpiggen-spørsmålet i dømet.

c) Det er ofte eit skjønnsspørsmål om ein treng get- og set-metodar og evt. andre metodar for å lese og endre tilstanden i eit objekt. Grunngje kva for slike metodar som trengst (om nokre) i **Question**-klassen og implementer dei.

d) **Quiz**-klassen representerer ein sekvens med spørsmål. Skriv kode for klassen med metodar for å:
- leggje til **Question**-objekt
- stille spørsmål med utskrift til skjerm (**System.out**) og lese inn svar frå tastaturet (**System.in**) heilt til det er svara riktig på kvart spørsmål, omtrent som vist i dømet i innleiinga.

e) Skriv naudsynte metodar for å kunne *køyre* **Quiz**-klassen (med nokre eksempelspørsmål) som eit hovudprogram etter reglane som Java har for dette. Når ein køyrer **Quiz**-klassen, skal altså brukaren bli stilt spørsmåla og få svare, omtrent som vist i dømet i innleiinga.

Del 3 – Input/output og unntak (IO) (10%)

Utvid **Quiz**-klassen med ein metode for å initialisere **Quiz**-objektet med spørsmål (og tilhøyrande svar og evt. svar-alternativ) frå fil. Fylgjande format skal støttast:
- Spørsmål, svar og evt. svar-alternativ har ei linje kvar.
- Spørsmålet kjem først, så svaret og deretter evt. svar-alternativ.
- Svar-alternativa skillast frå neste spørsmål med ei tom linje.

Eksempelfil som svarer til dømet i innleiinga, med kommentarar (som ikkje er en del av filinnhaldet) i høgre kolonne:

```
Kva heiter hovudstaden i Noreg?
Oslo

Kva slags ost er månen laga av?
Roquefort
Camembert
Roquefort
Brie

Kor høg er Galdhøpiggen?
2469
2469
2471

Er Java gøy?
ja
```

1. spørsmål
riktig svar
skiljelinje
2. spørsmål
riktig svar
1. svar-alternativ
2. svar-alternativ
3. svar-alternativ
skiljelinje
3. spørsmål
riktig svar
1. svar-alternativ
2. svar-alternativ
skiljelinje
4. spørsmål
riktig svar

Del 4 – Arv (15%)

I denne delen skal du nytta *arv* for å handtere ulike typar svar på en ryddigare måte. Det skal være ein klasse for frie tekst-svar kalla **StringQuestion**, ein klasse for tekst-svar med svar-alternativ kalla **StringOptionsQuestion**, og ein klasse for ja/nei-spørsmål kalla **BooleanQuestion**. Det som er felles for desse klassane, t.d. handtering av sjølve spørsmålsteksten, skal samlast i **Question**-superklassen. Bortsett frå at **Question**-klassen ikkje skal kunne instansierast, så skal *bruken* av den vere som i del 3, inkludert dei to **askQuestion**- og **checkAnswer**-metodane frå del 2. Prøv å strukturere klassene dine, så det blir *minst mogeleg duplisert kode*. Du står sjølvsagt fritt til å definere andre metodar som trengst i løysinga di.

Ein del av koden vil vere lik tidligare kode. Du kan sjølv velje om du vil skrive den på nytt, eller beskrive presist korleis tidligare skriven kode(tekst) kopierast inn i dei nye klassene.

a) Implementer først **Question**-superklassen og dei tre klassene **StringQuestion**, **StringOptionsQuestion** og **BooleanQuestion** med berre konstruktørar.

b) Implementer dei to metodane **askQuestion** og **checkAnswer**, slik at alle **Question**-objekt (eigentleg instansar av ein av dei tre andre klassene) i praksis verkar som i del 2.

c) Reimplementer metoden i **Quiz** for å lese inn spørsmål frå fil, slik at den verkar med dei nye **Question**-subklassene.

Del 5 – Trinnvis utførelse (15%)

I denne delen skal **Quiz**-klassen endrast slik at den held greie på tilstanden til ein runde med spørsmål og svar, men lar ein *annan* (hovudprogram)klasse styre framdrifta. Fylgjande tre metodar skal nyttast til å styre framdrifta:

- **start(boolean mode, PrintStream out, InputStream in)**: startar quiz-en (men ingen spørsmål stillast enno). **mode**-argumentet angir om eit spørsmål som det er svara galt på gjentakast *med ein gong* (**mode=false**), eller *etter at* alle etterfylgjande spørsmål er stilt

(**mode=true**). **out**-argumentet er straumen som spørsmål skal skrivast til (gjerast i **doQuestion**). **in**-argumentet er straumen som svara skal lesast frå (gjerast og i **doQuestion**).

- **doQuestion()**: Stiller eitt (neste) spørsmål og les eitt svar(forsøk). Kva for spørsmål som stillast avgjerast av kva for spørsmål som enno ikkje er stilt og svara riktig og **mode**-verdien som blei gjeven til **start**-metoden. Metoden returnerer talet på spørsmål som *enno ikkje er riktig svara på*.
- **stop()**: stopper quiz-en og returnerer kor mange spørsmål som *blei riktig svara på*.

Fylgjande kode eksemplifiserer korleis (denne versjonen av) **Quiz**-klassen er meint å bli nytta:

```
Quiz quiz = new Quiz();
// initialiser fra fil her (ikke vist)
quiz.start(true, System.out, System.in); // start quiz
while (quiz.doQuestion() > 0); // still spørsmål så lengje fleire står att
quiz.stop();
```

Implementer **start**-, **doQuestion**- og **stop**-metodane.

Del 6 – Grensesnitt (10%)

Det er ynskjeleg å kunne støtte ulike filformat for quiz-ar. Forklar med tekst og kode korleis du vil bruke Java-grensesnitt for å gjere det enkelt å bytte mellom ulike format og korleis koden du allereie har skriven for å lese inn frå fil kan utgjere (implementasjonen av) standardformatet. Merk at du ikkje skal implementere nye format, kun vise korleis grensesnitt-mekanismen gjer det enkelt.

Appendix

Fra Math-klassen:

`float java.lang.Math.signum(float f)`

Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.

Fra PrintStream-klassen:

`java.io.PrintStream`

A PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently.

`void java.io.PrintStream.print(String s)`

Prints a string. If the argument is null then the string "null" is printed. Otherwise, the string's characters are converted into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the write(int) method.

`void java.io.PrintStream.println(String x)`

Prints a String and then terminate the line. This method behaves as though it invokes print(String) and then println().