Høgskolen i Telemark

EKSAMEN

5609 OBJEKTORIENTERT PROGRAMMERING

14.12.2009

Tid:	4 timer, 9.00 – 13.00
Målform:	Bokmål / nynorsk (programkode er ikke oversatt)
Sidetall:	14
Hjelpemidler:	Alle trykte og skrevne (IKKE elektroniske)
Merknader:	Besvarelsen kan skrives med tekstprogram som ett dokument. **Lever besvarelsen i oppgaverekkefølge og skriv oppgavenummer foran hver delbesvarelse. Besvarelsen skal leveres som papirutskrift påført kandidatnr.**
Vedlegg:	1 side - Optional Attributes for HTML <input> tag

Eksamensresultatene blir offentliggjort på nettet. I tillegg finner du eksamensresultatslister på utsiden av eksamenskontoret, men da trenger du kandidatnummeret ditt. Du bør notere dette på en lapp som du tar vare på.

Besvarelsen kan skrives med et tekstbehandlingsprogram (Word, TextPad eller Notisblokk), uten kompileringsmulighet for Java. Fargekoding av Java syntaks er tillatt. Hele besvarelsen skal skrives som ett dokument (én fil) for å forenkle utskrift.

Besvarelsen skal leveres på papir, dvs. som utskrift fra skriver. Eksamensvaktene vil hjelpe til med utskrift. Du må selv se gjennom og kontrollere utskriften. Du er ansvarlig for at det som leveres inn er komplett. Figurer og tegningen som du har tegnet for hånd legges ved besvarelsen ved innlevering. Tid til utskrift og kontroll regnes ikke inn i eksamenstiden.

Bokmål

Oppgavesettet består av tre oppgaver om samme tema, som kan løses uavhengig av hverandre. Dvs: Selv om det er én oppgave du ikke har løst, kan du løse neste som om den forrige var løst. Disponer tiden godt, slik at du får gjort mest mulig på alle oppgavene. Det er viktigere å få fram program-logikken, enn at programmet er fritt for syntaksfeil og kjørbart. Om en oppgave er uklar så lag dine egne forutsetninger, og forklar disse.

Du skal lage noen Java-klasser som kan brukes i programmer for **on-line fotballtipping. NB! Du trenger ingen forkunnskap om fotballtipping for å løse denne oppgaven.**

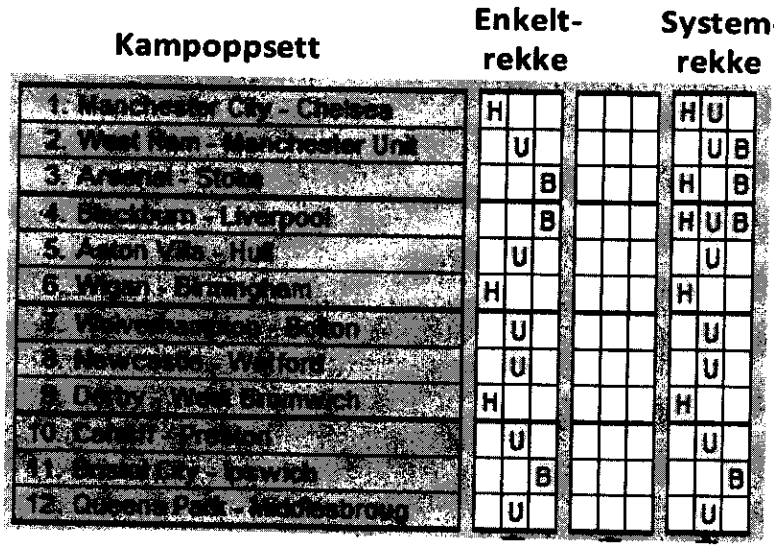
Faktarute – tipping:

En tippekupong består av et **kampoppsett** på 12 **kamper**. På hver kupong kan man tippe én eller flere **tipperekker**. I hver rekke tipper man resultatet av hver av de 12 kampene ved å fylle ut **tippetegn:** H i første rute for hjemmeseier, U i andre rute for uavgjort eller B i tredje rute for borteseier.

Det finnes to typer tipperekker: **enkeltrekker** og **systemrekker**. I en enkeltrekke kan man bare tippe ett tippetegn (resultat) for hver kamp. På en systemrekke kan man gardere, dvs. tippe flere tippetegn i samme kamp, ved å fylle ut 1, 2 eller 3 ruter i hver kamp.

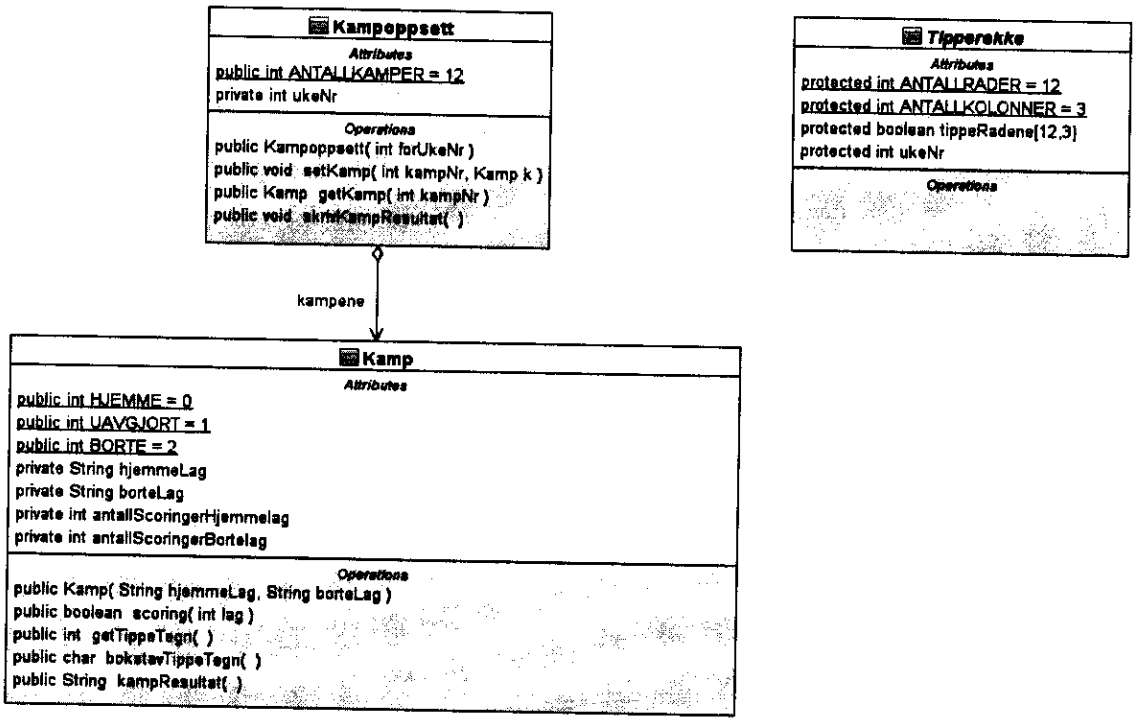
Pris
Det koster 1 kr å tippe én enkeltrekke. Prisen på en systemrekke avhenger av antall garderinger: Hver halvgardering (to tippetegn i samme rad) dobler prisen på rekken. Hver helgardering (tre tippetegn i en rad) tredobler prisen på rekken. Eksempel: Systemrekken til høyre i figuren nedenfor har tre halvgarderinger og én helgardering og koster altså: 1 kr * 2 * 2 * 2 * 3 = 24 kr.

En tippekupong med kampoppsett og eksempel på en enkeltrekke og en systemrekke, kan se slik ut:



Oppgave 1 (50 %)

UML klassediagrammet nedenfor viser **noen** av klassene i tippesystemet. Oppgavene nedenfor forklarer diagrammet og beskriver dessuten **flere metoder** i klassene enn det som er vist i diagrammet.



Oppgave 1a (10 %)

Programmer klassen Kamp i henhold til klassediagrammet i UML modellen, og beskrivelsen i punktene nedenfor. Du kan fritt gjøre dine egne utvidelser i klassen Kamp, men du skal ikke lage nye objektvariabler (instansvariabler). Metodene skal ha fornuftige feilsjekker.

- **Konstruktøren** skal brukes for å opprette en ny kamp med oppgitt navn på hjemmelag og bortelag, og sette stillingen til 0-0 ved "kampstart".
- Metoden **scoring** kan kalles for å oppdatere stillingen i kampen, hver gang et lag scorer et mål.
- Metoden **getTippeTegn** skal kunne kalles når som helst, og skal gi riktig tippetegn basert på resultatet i kampen (på det tidspunket metoden kalles). Tippetegnet skal returneres som en tallverdi slik:
  - Hjemme(seier) = 0
  - Uavgjort = 1
  - Borte(seier) = 2
- Metoden **bokstavTippeTegn** skal også returnere riktig tippetegn, men nå som én bokstav (H, U eller B).
- Metoden **kampResultat** skal returnere opplysninger om kampen og (nåværende) resultat som en tekststreng med dette formatet:

  Manchester U – Everton: 0-3 (B)

- Du kan utvide klassen med nye metoder hvis du får bruk for det senere i oppgavesettet.

Oppgave 1b (15 %)

Klassen **Tipperekke** skal brukes som en abstrakt superklasse for klasser som representer de rekkene spillerne har tippet (både **enkeltrekker** og **systemrekker**), og den riktige tipperekken (**fasitrekken**) etter at kampene er spilt.

Forklaring til UML diagrammet:

- Subklasser og metoder til klassen Tipperekke er ikke vist i UML-diagrammet!
- Objektvariabelen **tippeRadene** er en todimensjonal tabell som har én rad for hver kamp på tippekupongen (12), og 3 kolonner. Verdien *true* i kolonne 0 betyr hjemmeseier, *true* i kolonne 1 uavgjort, og *true* i kolonne 2 borteseier.
- For tipperekker som er tippet av en spiller, dvs. ikke fasitrekker, må hver rekke også inneholde et **spillernummer** (9-siffer) som identifiserer spilleren som har tippet rekken.

Oppgaver:

- Programmer klassen **Tipperekke** og nødvendige **subklasser** og **konstruktør(er)**.
- Lag en metode **setTippeTegn** som skal kunne kalles for å tippe (eller angi) resultatet i en kamp med tippetegn. For å "fylle ut" en tipperekke må vi kalle denne metoden (minst) en gang for hver kamp i rekken. Metoden må ta hensyn til reglene for enkelt- og systemrekker.
- Lag en metode **pris** som skal returnere prisen til en tipperekke i hele kroner. (Se faktarute på side 2.)
- Utvid klassene med andre metoder hvis du får bruk for det senere i oppgavesettet.

Oppgave 1c (5 %)

Lag et lite og enkelt testprogram som lager en "tilfeldig" tipperekke, dvs. som oppretter én enkeltrekke og som fyller den med et tilfeldig valgt tippetegn for hver kamp.

Oppgave 1d (10 %)

Starten på klassen **Kampoppsett** ser slik ut:

```
public class Kampoppsett {
    public final static int ANTALLKAMPER=12;
    private int ukeNr;
    private Kamp[] kampene; // Kamp nr.1 legges i rad nr.0 i tabellen

    .........
}
```

Programmér følgende metoder i klassen Kampoppsett. Se UML-diagrammet og husk feilkontroll i metodene!

- **Konstruktøren** skal opprette et (tomt) kampoppsett for et gitt ukenummer. (Tabellen med kamper skal fylles med kampdata senere i oppgave 2.)
- Metoden **setKamp** skal sette inn en kamp på rett plass i tabellen over kamper.
- Metode **getKamp** skal returnere kampen på et gitt kampnummer (plass i tabellen).
- Metoden **skrivKampResultater** skal skrive ut (til skjermen) alle kampene i kampoppsettet, med resultat. På utskriften skal kampene nummereres fra 1-12 i den rekkefølgen de ligger i tabellen.
- Lag en ny metode **getRettRekke** (ikke vist i UML-diagrammet), som returnerer en tipperekke med riktige tippetegn (fasitrekke), i forhold til kampresultatene på kupongen.

Oppgave 1e (5 %)

Programmer en ny metode **antallRette** som skal sammenlikne en tipperekke med en rekke som inneholder de riktige tippetegnene (fasitrekke), og returnere antall riktig tippede kamper. Metoden må fungere korrekt både for enkeltrekker og systemrekker.

Du må selv bestemme hvilken/hvilke klasse(r) metoden skal programmeres i.

Oppgave 1f (5 %)

Programmer klassemetoden finnGevinstRekker i programmet nedenfor:

```
import java.util.ArrayList;

public class TippeKontroll {

    // Rekker med 10 eller flere rette gir gevinst
    private static final int GEVINSTGRENSE = 10;

    private static Kampoppsett denneUkesKampoppsett;
    private static ArrayList<Tipperekke> innleverteTipperekker;
    private static ArrayList<Tipperekke> tippeRekkerMedGevinst;

    public static void main(String[] args) {
     denneUkesKampoppsett = new Kampoppsett(50);
     innleverteTipperekker = new ArrayList<Tipperekke>();

     .........
     // Kommentar 1: Se informasjon nedenfor

     tippeRekkerMedGevinst =
          TippeKontroll.finnGevinstRekker(denneUkesKampoppsett ,
                                          innleverteTipperekker);

     // Kommentar 2: Se informasjon nedenfor
    .........
    }

}
```

Kommentar 1: Her kan du forutsette følgende:

- at variabelen `denneUkesKampoppsett` inneholder denne ukens kampoppsett, med endelig kampresultat for alle kamper.
- at ArrayListen `innleverteTipperekker` inneholder alle innleverte tipperekker denne uken.

Kommentar 2: Her skal ArrayListen `tippeRekkerMedGevinst` inneholde alle innleverte tipperekker som resulterte i gevinst. Alle rekker med minst 10 rett tippede kamper gir gevinst. Hvilken type gevinst (10, 11 eller 12) er ikke interessant i denne oppgaven.

Oppgave 2 (20%)

I denne oppgaven skal du programmere en metode som kan lese kampdata fra en databasetabell. Tabellen heter **KAMPER** og er bygget i Oracle med følgende SQL kommando:

```
CREATE TABLE KAMPER(
  UKENR              INTEGER,
  KAMPNR             INTEGER,
  HJEMMELAG          VARCHAR2(40),
  BORTELAG           VARCHAR2(40),
  HJEMMESCORINGER    INTEGER,
  BORTESCORINGER     INTEGER,
  CONSTRAINT KAMPPK PRIMARY KEY (UKENR, KAMPNR)
);
```

Forklaring:

Kolonnene `ukeNr` og `kampNr` er en sammensatt primærnøkkel i tabellen KAMPER. Tabellen inneholder altså maksimalt én rad med samme ukenr og kampnr. Oracles datatype VARCHAR2 tilsvarer String i Java.

Oppgave:

Programmer ferdig klassemetoden lesKampoppsettFraDB i klassen TippeLagrer nedenfor, slik:

- Metoden skal lese data fra tabellen **KAMPER** for et oppgitt ukenummer, og returnere et **Kampoppsett** objekt med disse dataene. Hvis lesing feiler, eller det ikke finnes riktig antall kamper med data for angitt uke skal metoden returnere **`null`**.
- Lag evt. nye metoder i denne eller andre klasser dersom du har behov for det.

```
public class TippeLagrer {
  static private Connection forbindelse;

  static public Kampoppsett lesKampoppsettFraDB(int ukeNr) {...}

  public static boolean åpneDB() {
      try {
        Class.forName("oracle.jdbc.OracleDriver");
        forbindelse = DriverManager.getConnection(
      "jdbc:oracle:thin:@oraserver:1521:TIPPING", "user", "passwd");
        return (forbindelse != null);
      } catch (Exception e) { return false; }
  }

  public static boolean lukkDB()throws Exception {
    try {
       forbindelse.close();
        return true;
      } catch (Exception e) { return false; }
  }
}
```

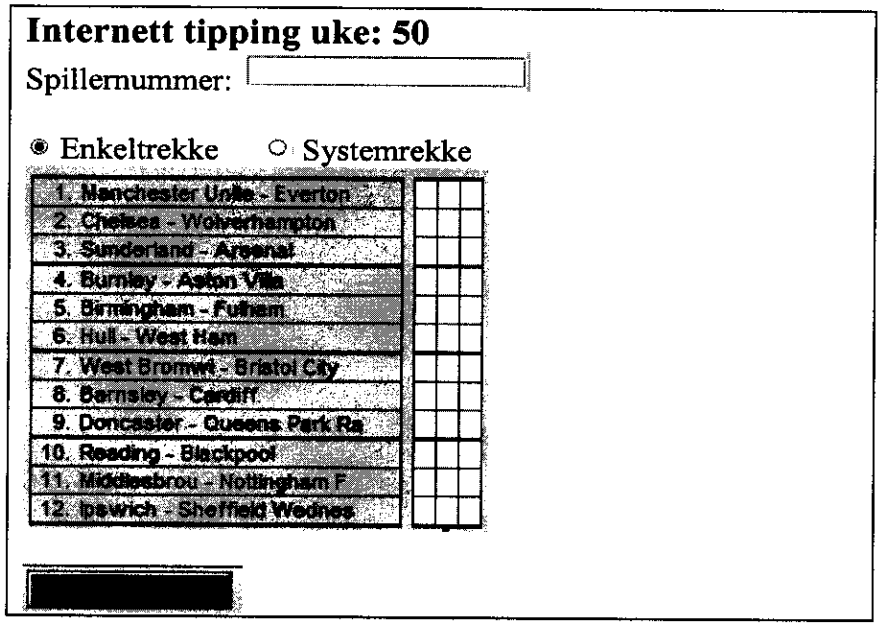
Oppgave 3 (30%)

I denne oppgaven kan du anta at alle klassene i tippesystemet fra oppgave 1 og 2 er programmert ferdig og tilgjengelig. Du kan derfor løse denne oppgaven selv om du ikke har løst oppgave 1 og 2.

Du skal nå lage en **webapplikasjon** i Java for å tippe on-line via Internett. Du kan løse oppgaven som en **JSP**, en **servlet** eller en kombinasjon av disse.

Ved hjelp av webapplikasjonen skal brukeren kunne fylle ut en tipperekke i systemet. For enkelhetsskyld skal det bare være mulig å tippe én rekke, dvs. enten en systemrekke eller en enkeltrekke.

Når webapplikasjonen kjøres skal den vise denne ukens tippekupong omtrent som vist nedenfor. Du behøver ikke kopiere dette utseende i detalj, og står fritt til å lage ditt eget utseende på websiden.



Krav til funksjonalitet i webapplikasjonen:

- Webskjemaet skal benytte data fra et **Kampoppsett** objekt, som skal opprettes ved hjelp av klassen **TippeLagrer** fra oppgave 2. (Du skal ikke programmere mot databasen i denne oppgaven.)
- Når webskjemaet vises må spilleren fylle ut sitt spillernummer og velge om han skal spille enkeltrekke eller systemrekke.
- Spilleren kan fylle ut kupongen ved å sette kryss i de rutene han vil tippe.
  - Tips: bruk en html-tabell med <input type = "checkbox".... > i hver rute.
  - Liste over attributter til <input> tag'en ligger som vedlegg bakerst i oppgavesettet.
  - Hvis du vil forenkle litt, kan tippetegnene evt. skrives som bokstaver i et tekstfelt.
- Når spilleren trykker **Levér kupongen**, skal webapplikasjonen **opprette et tipperadobjekt** basert på det utfylte webskjemaet. Deretter skal webapplikasjonen vise den ferdig utfylte tipperekka, med en bekreftelse på at rekken er mottatt, samt informasjon om pris på rekken.
- Du behøver ikke ta hensyn til at flere brukere kan benytte webapplikasjonen samtidig. Dvs. du skal ikke benytte sesjonshåndtering.

Slutt på oppgavesettet (bokmål)

Nynorsk

Oppgåvesamlinga består av tre oppgåver om same tema, som kan løysast uavhengig av kvarandre. Dvs: Sjølv om det er ei oppgåve du ikkje har løyst, kan du løyse neste som om den førre var løyst. Disponer tida godt, slik at du får gjort mest mogeleg på alle oppgåvene. Det er viktigare å få fram program-logikken, enn at programmet er fritt for syntaksfeil og kjørbart. Om ei oppgåve er uklar så lag dine eigne føresetnader, og forklar desse.

Du skal lage nokre Java-klasser som kan nyttast i programmer for **on-line fotballtipping. NB! Du treng ingen forkunnskaper om fotballtipping for å løyse denne oppgåva.**

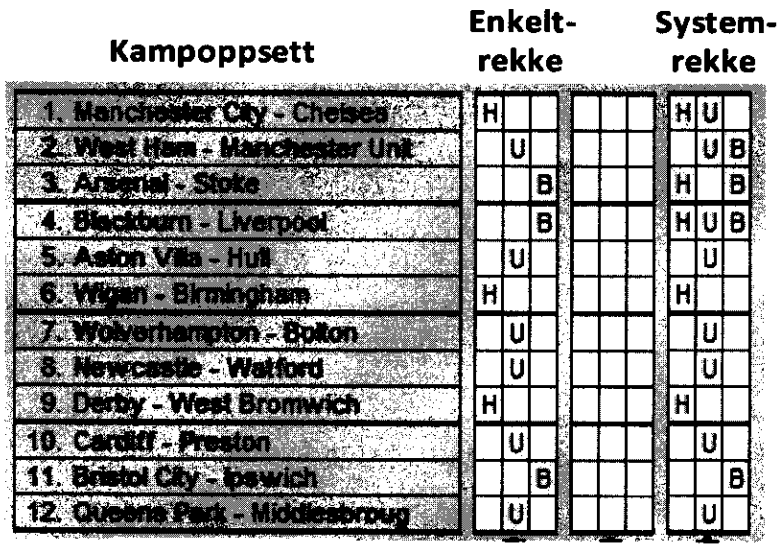
Faktarute – tipping:

Ein tippekupong syner eit **kampoppsett** på 12 **kampar**. På kvar kupong kan du tippe ein eller fleire **tipperekker**. I kvar rekke tipper du resultatet av kvar av dei 12 kampane ved å fylje ut **tippeteikn:** H i første rute for heimesiger, U i andre rute for uavgjort eller B i tredje rute for bortesiger.

Det finnes to typar tipperekker: **enkeltrekker** og **systemrekker**. I ei enkeltrekke kan du bare tippe eitt tippeteikn (resultat) for kvar kamp. I ei systemrekke kan du gardere, dvs. tippe fleire tippeteikn i same kamp, ved å fylje ut 1, 2 eller 3 ruter i kvar kamp.

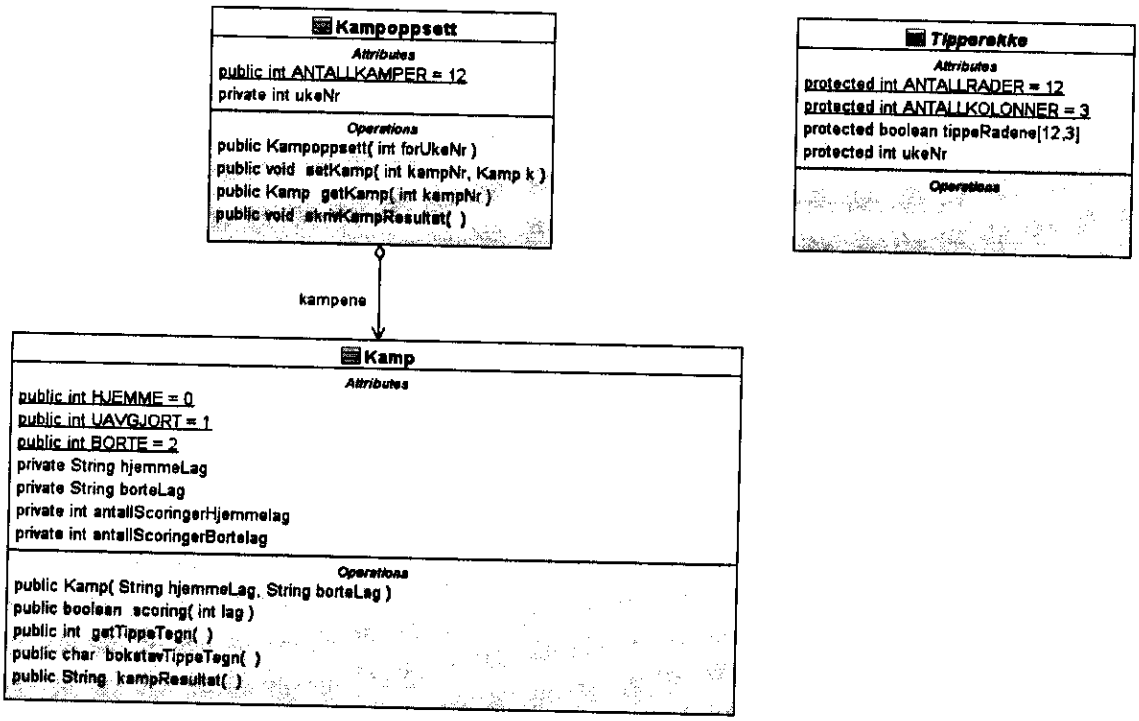
Pris
Det kostar 1 kr å tippe ei enkeltrekke. Prisen på ei systemrekke avheng av tal på garderingar: Kvar halvgardering (to tippeteikn i same rad) doblar prisen på rekka. Kvar heilgardering (tre tippeteikn i ei rad) tredoblar prisen på rekka. Døme: Systemrekka til høgre i figuren nedanfor har tre halvgarderingar og ein heilgardering og kostar altså: 1 kr * 2 * 2 * 2 * 3 = 24 kr.

Ein tippekupong med kampoppsett og døme på ei enkeltrekke og ei systemrekke, kan sjå slik ut:



Oppgåve 1 (50 %)

UML klassediagrammet nedanfor syner **nokre** av klassene i tippesystemet. Oppgåvene nedanfor forklarer diagrammet og omtalar dessutan **fleire metodar** i klassene enn det som er synt i diagrammet.



Oppgåve 1a (10 %)

Programmer klassa Kamp i samsvar med klassediagrammet i UML modellen, og omtala i punkta nedanfor. Du kan fritt gjere dine eigne utvidingar i klassa Kamp, men du skal ikkje lage nye objektvariablar (instansvariablar). Metodane skal ha fornuftige feilsjekkar.

- **Konstruktøren** skal nyttast for å opprette ein ny kamp med namn på heimelag og bortelag, og sette stillinga til 0-0 ved "kampstart".
- Metoden **scoring** kan kallast for å oppdatere stillinga i kampen, kvar gong eit lag scorar eit mål.
- Metoden **getTippeTegn** skal kunne kallast når som helst, og skal gi riktig tippeteikn basera på resultatet i kampen (på det tidspunktet metoden vert kalla). Tippeteiknet skal returnerast som ei talverdi slik:
  - Heime(siger) = 0
  - Uavgjort = 1
  - Borte(siger) = 2
- Metoden **bokstavTippeTegn** skal og returnere riktig tippeteikn, men no som ein bokstav (H, U eller B).
- Metoden **kampResultat** skal returnere opplysningar om kampen og (noverande) resultat som ein tekststreng med dette formatet:

  Manchester U – Everton: 0-3 (B)

- Du kan utvide klassa med nye metodar dersom du får bruk for det seinare i oppgåvene.

Oppgåve 1b (15 %)

Klassa **Tipperekke** skal nyttast som ei abstrakt superklasse for klasser som representer dei rekkene spillarane har tippa (både **enkeltrekker** og **systemrekker**), og den riktige tipperekka (**fasitrekka**) etter at kampane er spela.

Forklaring til UML diagrammet:

- Subklasser og metodar til klassa Tipperekke er ikkje synt i UML-diagrammet!
- Objektvariabelen **tippeRadene** er ein todimensjonal tabell som har ei rad for kvar kamp på tippekupongen (12), og 3 kolonnar. Verdien *true* i kolonne 0 tydar heimesiger, *true* i kolonne 1 uavgjort, og *true* i kolonne 2 bortesiger.
- For tipperekker som er tippa av ein spelar, dvs. ikkje fasitrekker, må kvar rekke og innehelde eit **spelarnummer** (9-siffer) som identifiserar spelaren som har tippa rekka.

Oppgåver:

- Programmer klassa **Tipperekke** og naudsynte **subklasser** og **konstruktør(ar)**.
- Lag ein metode **setTippeTegn** som skal kunne kallast for å tippe (eller angi) resultat i ein kamp med tippeteikn. For å "fylje ut" ei tipperekke må vi kalle denne metoden (minst) ein gong for kvar kamp i rekka. Metoden må ta omsyn til reglane for enkelt- og systemrekker.
- Lag ein metode **pris** som skal returnere prisen til ei tipperekke i heile kroner. (Sjå faktarute på side 8.)
- Utvid klassene med andre metodar dersom du får bruk for det seinare i oppgåvene.

Oppgåve 1c (5 %)

Lag eit lite og enkelt testprogram som lagar ei "tilfeldig" tipperekke, dvs. som oppretter ei enkeltrekke og som fyljer den med eit tilfeldig valt tippeteikn for kvar kamp.

Oppgåve 1d (10 %)

Starten på klassa **Kampoppsett** er slik:

```
public class Kampoppsett {
    public final static int ANTALLKAMPER=12;
    private int ukenr;
    private Kamp[] kampene; // Kamp nr.1 legges i rad nr.0 i tabellen

    .........
}
```

Programmer følgande metodar i klassa Kampoppsett. Sjå UML-diagrammet og hugs feilkontroll i metodane!

- **Konstruktøren** skal opprette eit (tomt) kampoppsett for eit gitt vekenummer. (Tabellen med kampar skal fyllast med kampdata seinare i oppgåve 2.)
- Metoden **setKamp** skal sette inn ein kamp på rett plass i tabellen over kampar.
- Metode **getKamp** skal returnere kampen på eit gitt kampnummer (plass i tabellen).
- Metoden **skrivKampResultater** skal skrive ut (til skjermen) alle kampane i kampoppsettet, med resultat. På utskrifta skal kampane nummererast frå 1-12 i den rekkefølja dei ligg i tabellen.
- Lag ein ny metode **getRettRekke** (ikkje synt i UML-diagrammet), som returnerar ei tipperekke med riktige tippeteikn (fasitrekke), i høve til kampresultata på kupongen.

Oppgåve 1e (5 %)

Programmer ein ny metode **antalRette** som skal samanlikne ei tipperekke med ei rekke som inneheld dei riktige tippeteikna (fasitrekke), og returnere tal på riktig tippa kampar. Metoden må fungere korrekt både for enkeltrekker og systemrekker.

Du må sjølv avgjere kva for klasse(r) metoden skal programmerast i.

Oppgåve 1f (5 %)

Programmer klassemetoden finnGevinstRekker i programmet nedanfor:

```
import java.util.ArrayList;

public class TippeKontroll {

    // Rekker med 10 eller fleire rette gir vinst
    private static final int GEVINSTGRENSE = 10;

    private static Kampoppsett denneUkesKampoppsett;
    private static ArrayList<Tipperekke> innleverteTipperekker;
    private static ArrayList<Tipperekke> tippeRekkerMedGevinst;

    public static void main(String[] args) {
     denneUkesKampoppsett = new Kampoppsett(50);
     innleverteTipperekker = new ArrayList<Tipperekke>();

     .........
     // Kommentar 1: Sjå informasjon nedanfor

     tippeRekkerMedGevinst =
          TippeKontroll.finnGevinstRekker(denneUkesKampoppsett ,
                                          innleverteTipperekker);

     // Kommentar 2: Sjå informasjon nedanfor
    .........
    }

}
```

Kommentar 1: Her kan du anta dette:

- at variabelen `denneUkesKampoppsett` inneheld denne vekas kampoppsett, med endeleg kampresultat for alle kampar.
- at ArrayList'a `innleverteTipperekker` inneheld alle innlevera tipperekker denne veka.

Kommentar 2: Her skal ArrayList'a `tippeRekkerMedGevinst` innehelde alle innlevera tipperekker som resultera i vinst. Alle rekker med minst 10 rett tippa kampar gir vinst. Kva for type vinst (10, 11 eller 12) er ikkje interessant i denne oppgåva.

Oppgåve 2 (20%)

I denne oppgåva skal du programmere ein metode som kan lese kampdata frå ein databasetabell. Tabellen heiter **KAMPAR** og er laga i Oracle med føljande SQL kommando:

```
CREATE TABLE KAMPAR(
  UKENR              INTEGER,
  KAMPNR             INTEGER,
  HJEMMELAG          VARCHAR2(40),
  BORTELAG           VARCHAR2(40),
  HJEMMESCORINGER    INTEGER,
  BORTESCORINGER     INTEGER,
  CONSTRAINT KAMPPK PRIMARY KEY (UKENR, KAMPNR)
);
```

Forklaring:

Kolonnane UKENR og KAMPNR er ein samansett primærnykkjel i tabellen KAMPAR. Tabellen inneheld altså maksimalt ei rad med same ukenr og kampnr. Oracles datatype VARCHAR2 svarar til String i Java.

Oppgåve:

Programmer ferdig klassemetoden lesKampoppsettFraDB i klassa TippeLagrer nedanfor, slik:

- Metoden skal lese data frå tabellen **KAMPAR** for eit gitt vekenummer, og returnere eit **Kampoppsett** objekt med desse data. Dersom lesing feilar, eller det ikkje finnes riktig tal på kampar med data for gitt veke skal metoden returnere **null**.
- Lag evt. nye metodar i denne eller andre klasser dersom du har behov for det.

```
public class TippeLagrer {
  static private Connection forbindelse;

  static public Kampoppsett lesKampoppsettFraDB(int ukenr) {...}

  public static boolean åpneDB() {
      try {
        Class.forName("oracle.jdbc.OracleDriver");
        forbindelse = DriverManager.getConnection(
      "jdbc:oracle:thin:@oraserver:1521:TIPPING", "user", "passwd");
        return (forbindelse != null);
      } catch (Exception e) { return false; }
  }

  public static boolean lukkDB()throws Exception {
    try {
       forbindelse.close();
        return true;
      } catch (Exception e) { return false; }
  }
}
```

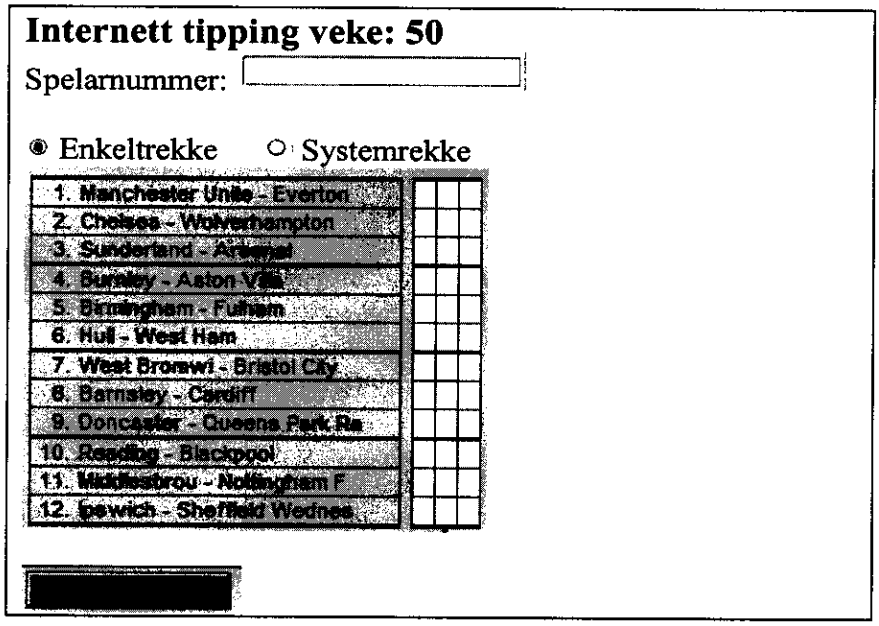
Oppgåve 3 (30%)

I denne oppgåva kan du anta at alle klassene i tippesystemet frå oppgåve 1 og 2 er programmera ferdig og tilgjengeleg. Du kan difor løyse denne oppgåva sjølv om du ikkje har løyst oppgåve 1 og 2.

Du skal no lage ein **webapplikasjon** i Java for å tippe on-line via Internett. Du kan løyse oppgåva med **JSP,** ein **servlet** eller ei kombinasjon av desse.

Ved hjelp av webapplikasjonen skal brukaren kunne fylje ut ei tipperekke i systemet. For å gjere oppgåva enklare skal det bare være mogeleg å tippe ei rekke, dvs. anten ei systemrekke eller ei enkeltrekke.

Når webapplikasjonen vert kjørt skal den syne denne vekas tippekupong omtrent som nedanfor. Du treng ikkje kopiere denne utsjånaden i detalj, og står fritt til å lage din eigen utsjånad på websida.



Krav til funksjonalitet i webapplikasjonen:

- Webskjemaet skal nytte data frå eit **Kampoppsett** objekt som skal opprettes ved hjelp av klassa **TippeLagrer** frå oppgåve 2. (Du skal ikkje programmere mot databasen i denne oppgåva.)
- Når webskjemaet vises må spelaren fylje ut sitt spelarnummer og velje om han skal spille enkeltrekke eller systemrekke.
- Spelaren kan fylje ut kupongen ved å sette kryss i dei rutene han vil tippe.
  - Tips: bruk ein html-tabell med <input type = "checkbox".... > i kvar rute.
  - Liste over attributt til <input> tag'en ligg som vedlegg på bak i oppgåvesamlinga.
  - Dersom du vil forenkle litt, kan tippeteikna evt. skrivast som bokstavar i eit tekstfelt.
- Når spelaren trykker **Levér kupongen**, skal webapplikasjonen **opprette eit tipperadobjekt** basera på det utfylde webskjemaet. Deretter skal webapplikasjonen syne den ferdig utfylde tipperekka, med ei stadfesting av at rekka er motteke, samt informasjon om pris på rekka.
- Du treng ikkje ta omsyn til at fleire brukarar kan nytte webapplikasjonen samstundes. Dvs. du skal ikkje nytte sesjonshandtering.

Slutt på oppgåvesamlinga (nynorsk)

Vedlegg: Optional Attributes for HTML <input> tag

DTD indicates in which HTML 4.01/XHTML 1.0 DTD the attribute is allowed. S=Strict, T=Transitional, and F=Frameset.

Attribute	Value	Description	DTD
accept	*MIME_type*	Specifies the types of files that can be submitted through a file upload (only for type="file")	STF
align	left right top middle bottom	Deprecated. Use styles instead. Specifies the alignment of an image input (only for type="image")	TF
alt	*text*	Specifies an alternate text for an image input (only for type="image")	STF
checked	checked	Specifies that an input element should be preselected when the page loads (for type="checkbox" or type="radio")	STF
disabled	disabled	Specifies that an input element should be disabled when the page loads	STF
maxlength	*number*	Specifies the maximum length (in characters) of an input field (for type="text" or type="password")	STF
name	*name*	Specifies a name for an input element	STF
readonly	readonly	Specifies that an input field should be read-only (for type="text" or type="password")	STF
size	*number*	Specifies the width of an input field	STF
src	*URL*	Specifies the URL to an image to display as a submit button	STF
type	button checkbox file hidden image password radio reset submit text	Specifies the type of an input element	STF
value	*value*	Specifies the value of an input element	STF