Tentamen i EDAF60

29 oktober 2018

Skrivtid: 14-19

- Skriv bara på ena sidan av pappret – tentorna kommer att scannas in, och endast framsidorna rättas.
- Skriv **INTE** med färgpenna – enda tillåtna färg är svart/blått/blyerts.
- Skriv tydligt – om texten inte går att läsa kan du inte få några poäng.
- Sätt identitet och sidnummer på varje inlämnat blad, kontrollera att sidnumret på din sista sida är samma som det antal blad du markerar på omslagspappret.

Poäng och betygssättning

Delpoängen på tentan räknas till antingen teori eller praktik (markerade med *T* eller *P* nedan), reglerna för betygen är:

Betyg 3: minst 60% på både teori- och praktikdel

Betyg 4: minst 70% på både teori- och praktikdel

Betyg 5: minst 80% på både teori- och praktikdel

Hjälpmedel:
- Pennor och blanka papper

Uppgift 1

Vi vill i denna uppgift att du kortfattat beskriver ett antal begrepp som förekommer i kursen. Vi kommer bara att läsa de 32 första orden i dina beskrivningar – den detaljnivå som vi är ute efter går att beskriva i färre ord än så.

(a) För var och en av SOLID-principerna: skriv först principens fullständiga namn, och därefter en beskrivning på högst 32 ord. *T 2.5 p*

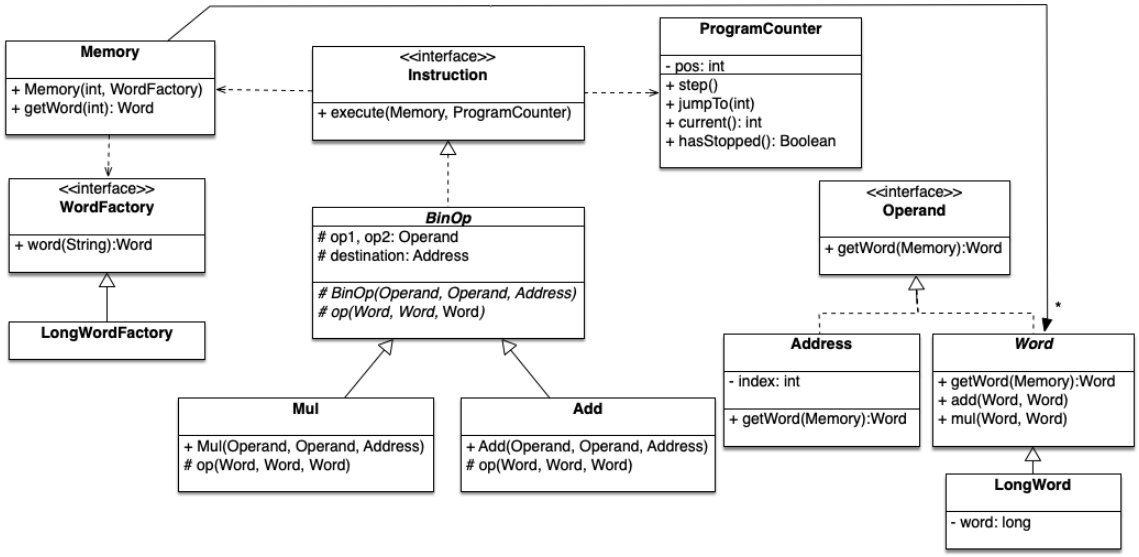
(b) Beskriv begreppet *cohesion* i högst 32 ord. En god design har _____ cohesion (skriv adjektivet vi söker i ditt svar). *T 0.5 p*

(c) Beskriv begreppet *coupling* i högst 32 ord. En god design har _____ coupling (skriv adjektivet vi söker i ditt svar). *T 0.5 p*

(d) Beskriv begreppet *DRY* i högst 32 ord. *T 0.5 p*

Uppgift 2

För en uppgift väldigt lik den ”Computer”-uppgift som ingår i kursen har en design med följande klassdiagram tagits fram.



En del av programmets Java-kod ser ni nedan:

```
Memory memory = new Memory(...);
ProgramCounter pc = new ProgramCounter();
Address n = new Address(4);
Instruction instruction = new Add(new LongWord(1), new LongWord(2), n);
instruction.execute(memory, pc);
```

Rita ett sekvensdiagram för `execute`-anropet på sista raden ovan. Utnyttja att ni vet att lösningen följer de mönster och principer ni lärt er i kursen.

T 3 p

Uppgift 3

I Computer-uppgiften kan vi ha ett `Word`-interface med följande specifikation:

```
interface Word {
    void add(Word left, Word right);
    void mul(Word left, Word right);
    void copy(Word other);
    boolean equals(Word other);
    String toString();
}
```

Med detta interface kan vi skriva $c = a + b$ på följande sätt (där vi antar att `a`, `b` och `c` alla är något slags `Word`-objekt som redan har skapats):

```
c.add(a, b);
```

För att skapa `Word`-objekt har vi:

```
interface WordFactory {
    Word word(String s);
}
```

Vi vill nu ha en `Word`-klass, `BooleanWord`, som representerar logiska värden – för dessa gäller:

- De kan ha det logiska värdet 'true' eller 'false'.
- Addition motsvarar logiskt 'eller' (så $a + b$ blir 'true' om a eller b är 'true').
- Multiplikation motsvarar logiskt 'och' (så $a \cdot b$ blir 'true' om både a och b är 'true').
- För att skapa ett `BooleanWord` med det logiska värdet 'false' skickar vi in strängen "0" till vår `BooleanWordFactory`, alla andra strängar representerar det logiska värdet 'true'.
- Metoden `toString()` returnerar strängen "`True`" om värdet är 'true', och "`False`" annars.

Klasserna `BooleanWord` och `BooleanWordFactory` skall ligga i samma paket.

(a) Implementera klassen `BooleanWord` (den måste implementera `Word` ovan). *P 3 p*

(b) Implementera klassen `BooleanWordFactory` (den måste implementera `WordFactory` ovan). *P 1 p*

Uppgift 4

(a) Skriv ett paket som implementerar Javas Observer/Observable-mönster. Du behöver inte kunna ta bort observers, inte hantera synkronisering, etc, det räcker att du implementerar de metoder i mönstret som du kommer att behöva använda i uppgift (b) nedan. Du behöver inte ha exakt rätt namn på alla metoder. *P 2 p*

(b) Vi har ett antal rörelsedetektorer och ett antal alarm, och vill att alla alarm skall starta om någon av detektorerna upptäcker att något rör sig.

Från början har vi klasserna

```
class MotionDetector {
    public void motionDetected(double distance, double angle) {
        // ... lägg in din egen kod här ...
    }
}
```

och

```
class Alarm {
    public void start() {
        // ... startar alarmet ...
    }
}
```

I `MotionDetector` kommer `motionDetected`-metoden att anropas automatiskt från hårdvara, så snart någon rörelse upptäcks – du får själv gärna lägga till kod i metoden. I `Alarm` kommer metoden `start()` att starta alarmet.

En tredje klass, `SurveillanceSystem`, används för att knyta samman detektorer och alarm, den skall ha åtminstone följande metoder för att registrera detektorer och alarm:

```
class SurveillanceSystem {
    public void add(MotionDetector detector) {
        // ... lägg in din egen kod här ...
    }
    public void add(Alarm alarm) {
        // ... lägg in din egen kod här ...
    }
}
```

De båda `add`-metoderna registrerar nya detektorer och alarm, de kan anropas i vilken ordning som helst.

Använd Observer/Observable-mönstret för att se till att alla alarm startar när en rörelse upptäcks av någon detektor – vi behöver inte bekymra oss för hur alarmen skall stängas av. Du får ändra fritt i klasserna ovan, låta dem utvidga klasser och implementera interfaces, men klasser och metoder som deklarerats skall finnas kvar och användas som beskrivet ovan.

För full poäng på uppgiften vill vi att du använder Observer-mönstret för hantera så mycket som möjligt av övervakningen, men du kan få poäng även om du 'fuskar' lite – gör så gott du kan. *P 3 p*

(c) Rita ett fullständigt UML-klassdiagram över samtliga klasser i lösningen till (b), inklusive dem i Observer/Observable-mönstret. *T 2 p*

Uppgift 5

Vi vill göra simuleringar, och vill kunna använda olika slags reella slumptal i dessa simuleringar – vi har följande interface för våra slumptalsgeneratorer:

```
interface RealRandom {
    double next();
}
```

och ett annat interface för simuleringarna (varje simulering använder bara ett slags slumptal):

```
interface Simulation {
    void run(RealRandom rng);
}
```

Vi kommer i uppgiften även att behöva använda en enkel databas med interfacet:

```
interface DB {
    void addSample(double sample);
}
```

(a) Implementera en klass `NormalRandom` som genererar normalfördelade slumptal med ett givet medelvärde och en given standardavvikelse, och en klass `UniformRandom` som genererar rektangelfördelade slumptal med givna min- och max-gränser. Båda klasserna skall implementera `RealRandom`-interfacet, och deras parametrar (medelvärde och standardavvikelse, eller min- och max-värde) skall anges när man skapar slumptalsgeneratorerna.

I standardklassen `Random` finns två användbara metoder:

```
class Random {
    double nextDouble();
    double nextGaussian();
}
```

För dessa gäller följande:

- `nextDouble()` ger ett rektangelfördelat tal r_u i intervallet $0 \leq r_u < 1$.
- `nextGaussian()` ger ett normalfördelat tal r_n med medelvärdet 0 och standardavvikelsen 1. Man kan göra om sådana till till ett givet medelvärde, μ, och en given standardavvikelse, σ, genom att multiplicera med standardavvikelsen och addera medelvärdet ($r_n \cdot \sigma + \mu$).

P 2 p

(b) Vi vill nu kunna studera de slumptal som genereras vid en simulering, *utan att vi behöver ändra någonting i slumptalsklasserna eller simuleringsklasserna*.

Vi vill i första hand kunna göra följande (inte nödvändigtvis båda samtidigt):

- beräkna medelvärde och standardavvikelse för de värden som använts vid en simulering, och
- spara alla värden som använts vid en simulering i en databas (`DB` i beskrivningen ovan).

Skriv klasser för att lösa dessa båda uppgifter, vi vill ha en klass för vardera uppgift – *använd lämpliga mönster från kursen*. Standardavvikelsen s för en följd $x_1, \ldots, x_n$ kan räknas ut som

$$s = \sqrt{\frac{\sum(x_i^2) - n \cdot m^2}{n - 1}},$$

där $\sum(x_i^2)$ är summan av kvadraterna på talen, och m är medelvärdet av talen.

Du får själv definiera lämpliga metoder i dina klasser, men de skall vara så enkla som möjligt att använda.

Skriv även programrader som kör simuleringen `BigBangSimulation` med rektangelfördelade slumptal i intervallet $0 \leq r < 4$, och sedan skriver ut medelvärde och standardavvikelse av de slumptal som använts i simuleringen. *P 3 p*

(c) Förklara vilka mönster från kursen du har använt i din lösning – förklara också var de förekommer i lösningen. *T 2 p*