Modellering av en dator

1 Syfte

Efter att gjort denna uppgift skall du kunna

- göra en UML-modell med Eclipse av ett system som beskrivs i en text och med ett testprogram.
- använda några designmönster.
- arbeta med Java-paket.
- implementera, exekvera och restrukturera Java-program med Eclipse.

2 Inledning

I denna uppgift skall du modellera en enkel dator och implementera modellen som ett Javaprogram. Modellen skall vara objektorienterad och baseras på denna beskrivning snarare än på hur en verklig dator är konstruerad.

Datorn har ett dataminne och plats för att ladda ett program. Uppdragsgivaren förväntar sig att modellen tillhandahåller

```
public class Computer {
    public Computer(Memory memory)
    public void load(Program program)
    public void run()
}
```

Dataminnet består av ord och varje ord finns på en adress i minnet. Minnets storlek bestäms när datormodellen skapas i huvudprogrammet. Varje ord kan användas för att lagra ett heltal. Olika instanser av datormodellen kan ha olika representation av ett ord. Vilken som skall användas bestäms när minnet skapas. Programmet är en lista av instruktioner. Det finns en programräknare som håller reda på index för nästa intruktion som skall exekveras. När programräknaren är negativ avbryts exekveringen.

Uppdragsgivaren kommer att använda följande testexempel som beräknar och skriver ut 5!.

```
public static void main(String[] args) {
    Program factorial = new Factorial();
    System.out.println(factorial);
    Computer computer = new Computer(new LongMemory(64));
    computer.load(factorial);
    computer.run();
}
```

```
public class Factorial extends Program {
    public Factorial() {
        Address n = new Address(0),
                fac = new Address(1);
        add(new Copy(new LongWord(5), n));
        add(new Copy(new LongWord(1), fac));
        add(new JumpEq(6, n, new LongWord(1)));
        add(new Mul(fac, n, fac));
        add(new Add(n, new LongWord(-1), n));
        add(new Jump(2));
        add(new Print(fac));
        add(new Halt());
    }
}
```

I exemplet representeras ord i minnet med klassen `LongWord` som använder typen `long` för att representera heltal. Eftersom konkurrenterna experimenterar med med datorer med dubbelt så långa ord vill uppdragsgivaren att klasserna skall utformas så att man kan lägga till klasser för `VeryLongWord` och `VeryLongMemory` utan att ändra på något utom `main` och `Factorial`. Dessa nya klasser skall ej implementeras. Det sista argumentet i konstruerarna för `Copy`, `Add` och `Mul` är alltid adresser. Alla argument till instruktionskonstruererarna utom heltalen i hopp-instruktionerna är antingen adresser till ord i minnet eller konstanter representerade som ord. Instruktionerna utför följande åtgärder.

- `Copy` kopierar det ord som anges av det första argumentet till den adress i minnet som finns i den andra argumentet.
- `Add` och `Mul` adderar resp. multiplicerar de ord som ges av de två första operanderna och lägger resultatet på den address som ges av det tredje argumentet.
- `Jump` sätter programräknaren till det instruktionsindex som finns i argumentet.
- `JumpEq` jämför de ord som ges av andra och tredje argumentet och sätter programräknaren till det instruktionsindex som finns i det första argumentet om de är lika.
- `Print` skriver ut det ord som anges av argumentet.
- `Halt` sätter programräknaren till ett negativt värde.

Testprogrammet ger en utskrift av instruktionerna i `factorial` och resultatet av exekveringen.

```
0 CPY 5 [0]
1 CPY 1 [1]
2 JEQ 6 [0] 1
3 MUL [1] [0] [1]
4 ADD [0] -1 [0]
5 JMP 2
6 PRT [1]
7 HLT

120
```

I utskriften av programmet är minnesadresser omgivna av hakparenteser.

3 Uppgift

Uppgiften skall lösas i grupper om fyra studenter. I undantagsfall kan kursansvarig lärare medge mindre gruppstorlek.

Uppgiften löses i två steg. I det första steget gör gruppen ett förslag till vilka paket, klasser, attribut och metoder som behövs för implementeringen och redovisar detta i form av klassdiagram. För varje paket skall det finnas ett klassdiagram som visar alla klasser som ni själva skall implementera med alla attribut och metoder som ni förutser behövs.

Inför arbetet i gruppen bör du fundera på följande.

1. Det finns ca 15 klasser och gränssnitt i programkoden ovan och det behövs ytterligare några för att kunna implementera programmet. Vilka klasser saknas och vilka är gränssnitt resp. klasser? Det finns ytterligare aritmetiska instruktioner och hoppinstruktioner med andra jämförelser än de som syns i testexemplet. Utformning och implementering av dessa ingår inte i uppgiften men designen skall göra det lätt att tillfoga dem senare.

2. Klassen `Program` tillhandahåller tydligen metoden `add` för att lägga till en instruktion till programmet. Om man låter `Program` utvidga en lämplig klass i `java.util` behöver man inte implementera metoden. Vilken klass bör man utvidga?

3. Klasserna skall fördelas på minst två paket. Vilka paket bör finnas och hur fördelas klasserna?

4. Studera designmönstret *Command* i kapitel 13 och hur det används i kapitel 18. Var och hur bör det användas i uppgiften.

5. Studera designmönstret *Template method* i kapitel 14 och användningen i kapitel 19. Mönstret skall användas för att undvika duplicerad kod i likartade klasser. Var kan detta bli aktuellt?

6. Studera designmönstret *Strategy* i kapitel 14. Hur använder man mönstret för att hantera `Address`- och `LongWord`-operander på ett enhetligt sätt?

7. När man exekverar `Add`-instruktionen skall man utföra en addition. I vilken klass skall additionen utföras?

8. Rita ett sekvensdiagram på papper, som visar alla inblandade objekt när `ADD`-kommandot i `Factorial` exekveras.

Det finns många verktyg (Computer Aided Software Engineering, CASE) som understödjer modellering, design, implementering, refactoring, testning och grupparbete. I denna kurs används Eclipse som är fri programvara och det bästa som finns för utveckling i Java. Klassdiagram skall konstrueras med valfritt Eclipse-baserat verktyg. Lägg inte tid på att försöka få diagrammen att se ut precis som ni eller läraren skulle önska. De skall bara användas för att underlätta designdiskussionen. Diagrammen skall visa alla egna klasser med alla attribut, metoder, generaliseringar och realiseringar.

Den elektroniska inlämningen skall innehålla klassdiagrammen som bilagor. Den skall göras senast ett dygn före designmötet. Sekvensdigrammet bifogas elektroniskt eller medföres vid mötet. Vid mötet vill läraren ha svar på frågorna. Diskussionen föranleder ofta en revidering av designen. Undermålig inlämning medför att mötet ställs in och att gruppen får inkomma med ett reviderat förslag.

I det andra steget implementerar gruppen programmet och redovisar genom elektronisk inlämning av reviderade UML-diagram och programkoden. Vidare skall nedlagd arbetstid för uppgiften redovisas för varje deltagare utan angivande av namn. Lärarens granskning kommer att fokusera på design och implementering. Därför är det viktigt att diagrammen innehåller all väsentlig information, att koden är välskriven och att klasser, metoder och attribut har väl valda namn. Inlämning skall ske senast söndagen den 26 september.

4 Inlämning

Inlämning av UML-diagram och programkod görs med e-post. Mottagare är `edaf10` (för båda kurserna), på domänen `cs.lth.se`.Subject-raden skall innehålla

`computer by` *user1 user2 user3 user4*

där *user1-4* är gruppmedlemmarnas användaridentiteter på efd-systemet. Breven hanteras på institutionen av ett program som heter Sam som distribuerar posten till lärarna på kursen. Ordet `computer` identifierar vilken uppgift det handlar om. Det är tillåtet att använda versaler. Ordet `by` avskiljer uppgiftsnamnet från listan med gruppmedlemmar och får utelämnas. Sam tolkar kommatecken och ord som det inte känner till som mellanslag. Lärarens svar går till alla gruppmedlemmar. Efterföljande korrespondens skickas till samma adress med samma titelrad.

UML-diagram bifogas som jpg eller pdf. Om du inte kan producera något av dessa format får du lämna diagram på papper till din handledare.

Programkod bifogas på motsvarande sätt genom att exportera `src`-katalogen som en zip-fil.