

Refaktorisering och UML med Eclipse

1. Introduktion

Eclipse är en utvecklingsmiljö för allehanda uppgifter. Den innehåller bland annat ett mycket kvalificerat stöd för programutveckling i Java. Den är särskilt lämplig för denna kurs eftersom det är lätt att göra designförändringar och det finns integrerade verktyg för design med UML.

Eclipse är utformat som ett ramverk så att man enkelt kan lägga till ny funktionalitet med ”stickproppar” (plug-ins) som man kan hitta på nätet eller skriva själv. Propparna innehåller i regel kompillerade Java-klasser som implementerar gränssnitt i ramverket. Även Java-miljön ingår i systemet som stickproppar. I denna laboration skall du installera en stickpropp från nätet för att kunna modellera med hjälp av UML.

Det kommer löpande nya versioner av Eclipse och proppar som inte alltid fungerar tillsammans.

Eclipse är implementerat i Java och kan därmed exekveras på olika plattformar och finns både på Linux- och Windows-maskinerna i EFD. Användarens konfigurationsfiler håller reda på var Java Runtime Environment finns på datorn vilket kräver omkonfiguration om man byter plattform i EFD-systemet. Du bör inledningsvis använda Linux.

2. Konfiguration

Programmet startas via applikationsmenyn (**Applications** → **Programming** → **Eclipse**).

Det är lämpligt att skapa ett nytt arbetsområde (*workspace*) för denna kurs. När du startar Eclipse får du eventuellt tillfälle att välja arbetsområde i en dialogruta. Om så inte är fallet öppnar du dialogrutan via menyvalet **File** → **Switch Workspace**. Ange en sökväg som t.ex. slutar med omd/workspace. Eclipse skapar de kataloger som inte finns.

Sedan öppnas en välkomstsida som innehåller ikoner som leder till omfattande dokumentation, handledningar och nyheter. Om du inte använt Eclipse tidigare ger steg-för-steg-handledningen bakom ikonerna **Tutorials** → **Create a Hello World Application** en kort introduktion.

Det finns ett många Eclipse-baserade verktyg för att modellera med UML, se t.ex. <http://marketplace.eclipse.org/>. Det verktyg vi använt tidigare är finns inte längre i någon fri version så vi måste byta. Jag vet inte vilket som är bäst och studenter brukar ha olika preferenser. Beskrivningen i detta dokument använder Green UML, <http://green.sourceforge.net>. Green UML är tämligen primitivt och kan bara användas för klassdiagram. Förhoppningsvis duger det för användning i denna kurs, men knappast i kommersiella sammanhang. Det är tillåtet att använda valfritt datorbaserat verktyg för att producera UML-diagram. Om någon hittar något som är riktigt bra vill jag gärna bli informerad.

Du skall själv konfigurera Eclipse för att kunna använda UML-verktyget. Under menyalternativet **Help** → **Install new software** skall du lägga till två plugins. Välj **Add** fyller i Location med <http://download.eclipse.org/tools/gef/updates/releases>. Följ anvisningarna tills installationen är klar. Välj den senaste releasen. Det tar flera minuter att installera denna plugin. Det är förmodligen onödigt att starta om Eclipse innan man installerat nästa plugin.

Gör sedan samma sak med <http://www.cse.buffalo.edu/faculty/alphonse/green>. Välj att installera Green men inte Green Add-Ons

Det är lämpligt att ändra inställningarna för Green UML under **Preferences** → **Green** åtminstone så att bara **Generate stubs for inherited abstract methods** och **Save compilation units** är markerade. I **Preferences** → **Green** → **Class Boxes** bör **Show method parameter names** och **Show tooltips** vara markerade.

3. Java-editorn

Standard-editorn för Java kan väldigt mycket av språket och rapporterar hela tiden fel och är beredd att hjälpa till på olika sätt. Fel markeras med vågstrecksunderstrykning och ett erbjudande om hjälp indikeras med en liten glödlampa i vänsterkanten.

Om man skriver början på ett reserverat ord eller ett namn och därefter CTRL-mellanslag så kommer editorn att föreslå ett antal alternativ. Efter "wh" ges fyra alternativ för while-satser och några klassnamn som du förmodligen inte hört talas om. Genom att flytta sig till alternativen kan man få se hur Eclipse kommer att komplettera. Man accepterar med return eller dubbelklick. Editorn känner också till namn på dina egna variabler, metoder och klasser.

Man kan skriva main och få en hel main-metod och sysout för System.out.println(). Ett tos på en plats där man kan definiera metoder ger som första alternativ att skugga metoden toString. Mallarna finns i **Preferences → Java → Editor → Templates** där man själv kan modifiera och lägga till.

När man skriver en punkt efter ett objektnamn erbjuds också ett antal alternativ. Antalet reduceras allteftersom man skriver vidare.

Om man skriver x=1; utan att ha deklarerat x visar editorn en liten lampa som erbjuder att deklarera en lokal variabel, lägga till ett attribut eller en parameter. När lampan är dekorerad med ett litet kryss finns det fel som måste åtgärdas. Ett utropstecken betyder ofta att något är onödigt och kan tas bort utan att programmets funktionalitet förändras.

4. Menyer och genvägar

Det finns ofta flera sätt att anropa en operation.

De flesta operationerna kan nås via huvudmenyerna, **File**, **Edit**, etc. Många av operationerna förutsätter att man markerat det element som skall påverkas. Om man inte gjort kan alternativet vara nedtonat eller saknas.

En del operationer har egna ikoner under raden med huvudmenyer.

Editorer och vyer har ofta egna verktygsikoner.

Genom att högerklicka på ett element öppnar man elementets kontextmeny. Om man vill att operationen skall gälla alla element i editorn skall man se till att inget element är markerat och högerklicka i bakgrunden.

En del kommandon har tangentbordsgenvägar. Sådana syns till höger i vanliga menyer.

Mycket går att konfigurera efter eget tycke, t ex via **Preferences → General → Keys och Window → Customize Perspective**. . . . Inställningar kan sparas, exporteras och importeras.

5. Source-menyn

Med **Source → Format** formateras programmet så att indenteringen, inskjutna mellanslag, tomma rader etc. blir konsekvent. Det går att i detalj styra hur detta skall gå till i **Preferences → Java → Codestyle → Formatter** om man inte är nöjd med standardvärdena.

Medlemmarna i en klass (attribut, metoder, klasser etc.) bör vara ordnade på samma sätt i alla klasser i ett projekt. Med **Source → Sort members** åstadkommer man detta i en klass, ett paket eller i hela projektet. Via **Preferences → Java → Appearance → Member Sort Order** kan man styra detaljer för sorteringen.

När man behöver en ny konstruerare som ger värden åt några attribut använder man med fördel **Source → Generate Constructor using Fields**.

Source → Organize imports sorterar import-klausuler, tar bort onödiga och lägger till sådana som behövs.

Med **Source → Clean Up** . . . får man en mer genomgripande städning av programmet med indentering, sortering, mm. Detaljerna kan styras via **Preferences → Java → Code Style → Clean Up**.

6. Refactor-menyn

Refactor-menyn innehåller mängd operationer för att omstrukturera programmet. Innehållet i menyn växlar efter vad som är markerat i Package Explorer eller editorer.

Namnbyten, **Refactor → Rename** ..., är en enkel och kraftfull operation. Om man byter namn på en klass kommer alla referenser till namnet i hela projektet att bytas ut. Om man så önskar kommer även liknande namn på attribut och metoder och förekomster i kommentarer att modifieras.

Det enklaste sättet att flytta klasser mellan olika paket är att dra dem i Package Explorer. Det går att flytta metoder och attribut på samma sätt, men det kräver ofta efterarbete.

I uppgiften nedan får du prova några andra operationer. Experimentera gärna; i regel går det att återställa programmet med **Edit → Undo**.

7. Green UML

UML-modellen är integrerad med Java-representationen av programmet så att förändringar i den ena i regel avspeglar sig i den andra. Java-filer måste sparas för att detta skall hända. Om UML-modellen inte är uppdaterad finns kommandot **Refresh Editor** i bildens kontextmeny eller kortkommandot **f**.

Det kan vara bekvämt att skapa klasser och gränssnitt med UML-editorns palett, men undvik att skapa associationer; verktyget skapar de associerade objekten med parameterlösa konstruerare. Detta är nästan alltid fel.


Programmet saknar automatisk layout-funktion. Det är inte alltid så lätt att få diagrammet att se ut exakt som man skulle önska. Ansträng dig inte mer än att diagrammet blir användbart som underlag för designdiskussionen.


8. Uppgift

Denna uppgift innehåller enkel objektorienterad design och introducerar de viktigaste handgreppen i Eclipse.


1. Starta Eclipse med ett nytt arbetsområde (workspace) för denna kurs.

2. Installera och konfigurera UML-verktyget.

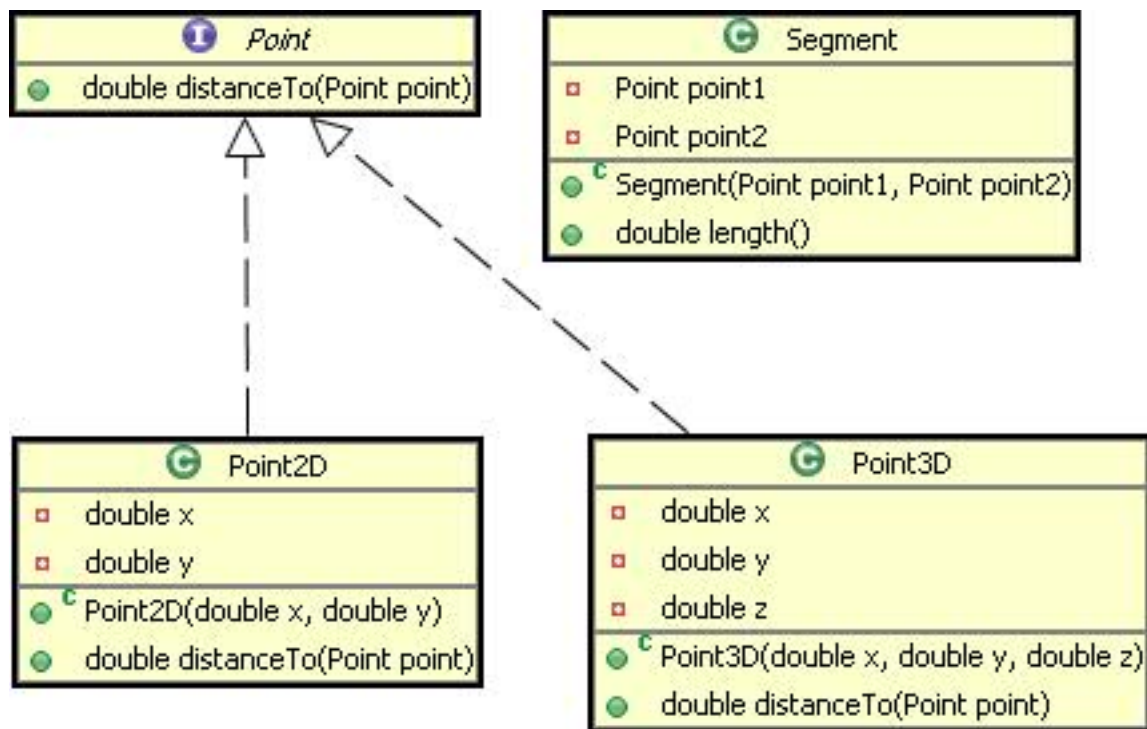
3. Skapa ett nytt Java-projekt (geometry) via. 

4. Skapa ett nytt Java-paket (element) med . 

5. Öppna ett nytt UML-diagram med hjälp av kontextmenyn för paketet genom att välja **Green UML → Add to New Class Diagram**. Motsvarande fil får namn efter projektet. Det kan vara lämpligt att namnge det efter paketet för att underlätta när det blir flera paket. **File → Save As**

6. Skapa klassen Point i UML-diagrammet genom att markera  **Class** i UML-paletten och klicka på den plats i diagrammet där klassrutan skall placeras. "Source folder" (geometry/src) och "Package" (element) måste anges.
Motsvarande Java-klass dyker upp i Package Explorer.

7. Öppna klassen och titta på koden. Skriv in en JavaDoc-kommentar.
8. Lägg till två privata attribut, `x` och `y`, av typen `double` i Java-editorn. När man sparar filen kommer attributen att bli synliga i UML-diagrammet.
9. Skapa en konstruktor med de två attributen via **Source** → **Generate Constructor Using Fields**. Kryssa gärna i **Omit call to default constructor super()**.
10. Lägg till metoden `public double distanceTo(Point other)` i Java-editorn och låt den returnera avståndet mellan `this` och `other`. Gör en JavaDoc-kommentar för metoden.
11. Formattera klassen med **Source** → **Format**.
12. Gör en JUnit-test för `distanceTo` via **File** → **New** → **JUnit Test Case**. Följ anvisningen att lägga till JUnit-paketet i "build path". Implementera metoden "test" och använd någon assert-metod. Exekvera testet med **Run** → **Run As** → **JUnit Test**.
13. Skapa ett nytt paket för tester och flytta testklassen dit. Testa igen.
14. Vi skall nu ändra modellen så att den kan hantera punkter och segment antingen i 2 eller 3 dimensioner. Byt namn på `Point` till `Point2D` via **Refactor** → **Rename**.
15. Extrahera ett gränssnitt från `Point2D` med metoden `distanceTo` via **Refactor** → **Extract Interface**. Låt gränssnittet få namnet `Point`.
16. Ändra signaturen för `distanceTo` med **Refactor** → **Change Method Signature** så att argumentet får typen `Point`. Korrigera `Point2D` så att `distanceTo` fortfarande fungerar.
17. Titta på UML-diagrammet och se så att det har blivit uppdaterat.
18. Skapa en klass `Point3D` med tre koordinater genom att utgå från en kopia av `Point2D`.
19. Infoga `Point3D` i UML-diagrammet genom att välja **Green UML** → **Add to Last Viewed Class Diagram** i kontextmenyn i "Types"-fönstret.
20. Skapa en klass, `Segment`, som representerar en sträcka med hjälp av dess ändpunkter.
21. Implementera metoden `double length()` i `Segment` så att den returnerar sträckans längd.
22. Testa metoden.
23. Använd **Source** → **Clean Up**... på hela paketet. Markera **Use costum profile** och konfigurera efter egna önskemål, t ex genom att avmarkera **Add missing Annotations** under fliken **Missing Code** och markera de fem rutorna under fliken **Code Organizing**.
24. Under Linux går det för närvarande inte att göra utskrifter direkt från Eclipse. Detta är ett pinsamt fel som funnits alltför länge. Java-kod skriver man lämpligen ut via ett terminalfönster i källkodskatalogen. UML-diagram går att exportera på tre olika former: jpg, gif och png, via kontextmenyn **Save As**. Prova vilket format som fungerar bäst.
25. Orientera dig om vad som kan konfigureras i **Preferences**.
26. När uppgiften är färdig bör UML-diagrammet se ut ungefär så här.



9. Redovisning

Ingen redovisning skall lämnas för laborationen och det är möjligt att utföra den på egen hand.