Vägledning till BlueJ

Michael Kölling
översättning Set Lonnert
PLATSHÅLLARE
ERSÄTT FÖRSTA SIDAN MED ORGINAL

13 februari 2001

Innehåll

Figurer

Kapitel 1

Förord

1.1 Om BlueJ

Denna vägledning[1] är en introduktion för att använda programmeringsmiljön BlueJ . BlueJ är en utvecklingsmiljö för Java™ speciellt designad för undervisning på en introducerande nivå. Den är utvecklad och implementerad av teamet för BlueJ vid Monash University, Melbourne, Australien. Mer information om BlueJ finns på `http://bluej.monash.edu`.

1.2 Räckvidd och läsare

Denna vägledning är avsedd för de som vill bekanta sig med möjligheterna hos miljön. Den förklarar inte designbeslut eller de forskningsfrågor som finns bakom miljön.

Det förutsätts att läsaren är bekant med programmeringsspråket Java — inget försök görs att ge en introduktion till Java i denna vägledning.

Detta är ingen fullständig referensmanual till miljön. Många detaljer har utelämnats — betoningen ligger på en kort och koncis introduktion, snarare än en komplett förteckning av funktioner.

De flesta avsnitt slutar med en summerande slutsats för avsnittet. Kapitel 10 repeterar alla summeringar som snabbreferens.

1.3 Copyright, licensering och distribution

1.4 Återkoppling

Kommentarer, frågor, korrigeringar, kritik och annan typ av återkoppling gällande systemet BlueJ eller denna vägledning är varmt välkomna och uppmuntras aktivt. Vänligen skicka

[1] Översättning Set Lonnert

epost till Michael Kölling (mik@monash.edu.au).

Kapitel 2

Att sätta igång

2.1 Installation

BlueJ distribueras som ett arkiv av Javaklasser i ”jar” format. Installering är ganska lätt.

2.1.1 Förutsättningar

Du måste ha JDK 1.2.2 eller senare installerat på ditt system för att använda BlueJ . Vissa delar fungerar bättre med JDK 1.3, så det är värt att hämta eller uppdatera till senaste utgåvan av JDK. Om du inte har JDK installerat kan du ladda ner det ifrån Suns webbplats `http://java.sun.com/j2se/`.

2.1.2 Att hämta BlueJ

BlueJ distribueras som fil kallad *bluej-xxx.jar*, där *xxx* är versionsnummer. Till exempel har BlueJ version 1.1.1 distributionsnamnet *bluej-111.jar*. Du kan få denna fil via diskett eller ladda ner det från webbplatsen för BlueJ `http://bluej.monash.edu`.

2.1.3 Om SDK, JDK och JRE

Ibland finns förvirring över olika distributioner av Java: SDK, JDK och JRE paketen. Du bör installera senaste versionen av Java 2 SDK (Software Development Kit). Termen JDK (Java Development Kit) är ett äldre namn för samma sak. Sun har ändrat namnkonventionen vid ett tillfälle, men ibland används fortfarande det äldre namnet (JDK). Till exempel, när du installerar Java 2 SDK v. 1.3, så kommer standardinstallationen att medföra biblioteksnamnet jdk1.3.

JRE (Java Runtime Enviroment) är annorlunda: Det är en delmängd av SDK för körning av Java. För BlueJ räcker inte detta. Vi behöver SDK därför att det inkluderar några utvecklingsverktyg som BlueJ behöver. JRE installeras automatiskt som del av installationen av SDK.

2.1.4 Installation

Windows:

Dubbelklicka på installationsfilen (*bluej-xxx.jar*). Om ditt system inte är konfigurerat för exekverbara jar-filer, kanske inte dubbelklickandet fungerar. Om det är så, öppna en fönster

MS-DOS Prompt och följ instruktionerna för Unix.

Unix:

Kör installeraren genom att exekvera följande kommando. OBS: För detta exempel använder jag distributionen i filen *bluej-111.jar* — du behöver använda namnet på den fil du har (med det rätta versionsnumret).

```
<jdk−sökväg>/bin/java −jar bluej−111.jar
```

`<jdk-sökväg>` är den katalog där JDK har installerats.

Ett fönster kommer att dyka upp som låter dig välja katalogen där BlueJ installeras och den version av JDK som kommer att köras ihop med BlueJ . Viktigt: Sökvägen till BlueJ (dvs. vilken som helst av föräldrakatalogerna) får inte innehålla mellanslag (t.ex. "Program Files")!

Klicka på Install. Efter avslutning kommer BlueJ vara installerat.

Om du har problem, se den FAQ som finns på webbstället för BlueJ.

2.2 Att starta BlueJ

Installationen av BlueJ installerar ett skript bluej i installationsbiblioteket. Från en GUI-miljö kan du bara dubbelklicka filen. Från kommandoraden (t.ex. Unix eller DOS), kan du starta BlueJ med eller utan ett projekt som argument:

```
$: bluej
```

eller

```
$: bluej examples/people
```

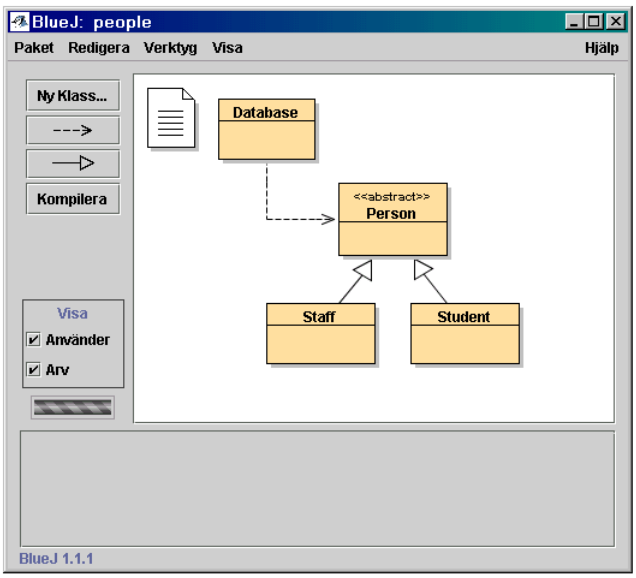
2.3 Öppna ett projekt

Projekten i BlueJ är liksom standardpaketen i Java, bibliotek som innehåller filerna i projektet. Om du startar BlueJ från kommandoraden och ger ett projekt som argument, kommer det att öppnas automatiskt. Om du startar BlueJ utan argument, använder du Paket - Öppna... i menyn för att välja och öppna projektet.

Kapitel 3

Grunderna — redigera / kompilera / exekvera

För detta handboksavsnitt, öppnar du projektet *people* som är inkluderat i distributionen av BlueJ . Du kan hitta det i biblioteket *examples* i hembiblioteket för textsfBlueJ . Efter öppnandet av paketet bör du se något liknande det fönster du ser i Figur 3.1. Fönstret behöver inte se exakt likadant ut som på ditt system, men skillnaderna bör vara små.



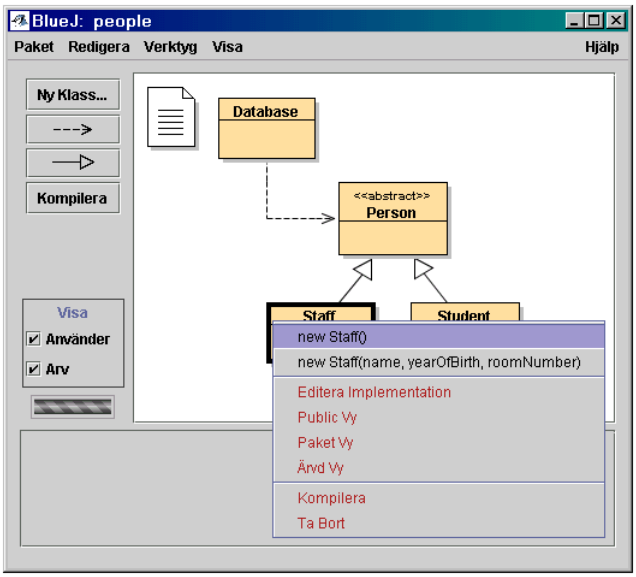
Figur 3.1: Huvudfönstret i BlueJ

3.1 Skapa objekt

En av de fundamentala egenskaperna för BlueJ är att du inte bara kan köra hela tillämpningar, men också direkt interagera med enskilda objekt från vilken som helst klass och köra de-

ras publika metoder. En exekvering i BlueJ görs vanligtvis genom att skapa ett objekt och sedan anropa en av objektets metoder. Detta är mycket användbart under utveckling av en tillämpning — du kan testa klasserna individuellt omedelbart efter de har skrivits. Hela tillämpningen behöver inte skrivas först.[1]

Rutorna som du ser i mitten av huvudfönstret (kallad *Database*, *Person*, *Staff* och *Student*) är ikoner representerande klasser i denna tillämpning. Du kan få en meny med operationer tillämpbara på klassen genom att högerklicka på klassikonen (Figur 3.2). Operationerna som visas är new operationer för varje konstruktor definierad för denna klass (först) följt av några operationer givna av omgivningen.



Figur 3.2: Klassoperationer (popupmeny)

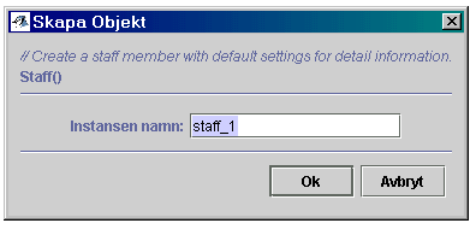
Vi vill skapa ett objekt från *Staff* (anställd), så du bör högerklicka på ikonen *Staff* (så poppar menyn upp som visas i Figur 3.2). Menyn visar två konstruktorer för att skapa ett objekt av *Staff*, en med parametrar och en utan. Välj först konstruktorn utan parametrar. Dialogen i Figur 3.3 visas.

Dialogen frågar efter ett namn för det objekt som skall skapas. Samtidigt föreslås ett namn (*staff_1*). Det räcker gott med standardnamnet för tillfället, så klicka bara OK. Ett objekt från *Staff* kommer att skapas.

Sedan objektet har skapats placeras det på objektbänken (Figur 3.4). Detta är allt som behövs för objektskapande: välj en konstruktor från klassmenyn, exekvera den och du har ett objekt placerat på objektbänken.

Du har kanske lagt märke till att klassen *Person* är etiketterat «`abstract`» (det är en abstrakt klass). Du kommer att märka (om du prövar) att du inte kan skapa objekt från abstrakta klasser (som specifikationen för Java definierar).

[1] Statiska metoder kan exekveras direkt utan att skapa objektet först. En av de statiska metoderna kan vara "*main*", så vi kan göra samma sak som händer i vanliga Javatillämpningar — starta en tillämpning bara genom att exekvera en statisk main-metod. Vi kommer att återkomma till detta senare. Men vi kommer att göra en del intressantare saker först, som inte vanligtvis kan göras i Javamiljöer.



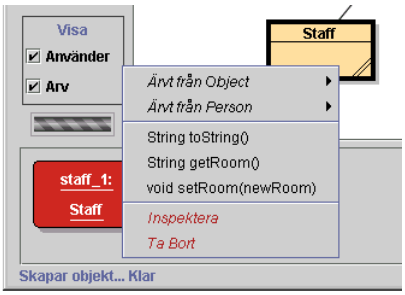
Figur 3.3: Skapandet av objekt utan parametrar

Figur 3.4: Ett objekt på objektbänken

Summering: För att skapa ett objekt, välj en konstruktor från klassens popupmeny.

3.2 Exekvering

När du nu har skapat ett objekt, kan du exekvera dess publika operationer. Högerklicka på objektet och en meny med objektoperationer kommer att poppa upp (Figur 3.5). Menyn visar tillgängliga metoder för detta objekt och två speciella operationer givna av omgivningen (Inspektera och Ta bort). Vi kommer att diskutera dessa senare. Låt oss först koncentrera oss på metoder.



Figur 3.5: Objektmenyn

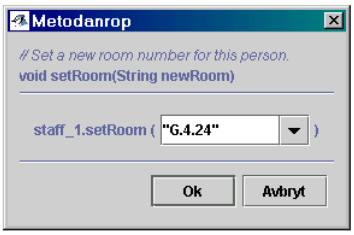
Du ser att det finns metoderna *getRoom* och *setRoom* vilka sätter och returnerar rummets nummer för de anställda (*staff*). Försök anropa *getRoom*. Välj det från objektmenyn så

kommer det exekveras. En dialog uppenbaras som visar dig resultatet av anropet (Figur 3.6). I detta fall säger namnet ”(unknown room)”, okänt rum, därför att vi aldrig specificerade rummet för denna person.

Figur 3.6: Visande av funktionens resultat

Metoder som ärvs från en superklass är tillgängliga genom en undermeny. Vid toppen av objektets popupmeny finns två undermenyer, en för metoder som ärvs från *Object* och en från *Person* (Figur 3.6). Du kan anropa metoderna i *Person* (som *getName*) genom att välja dom från undermenyn. Försök. Du kommer att se att svaret är lika vagt: det säger ”(unknown name)”, okänt namn, därför att vi har inte angivit vår persons namn.

Försök nu att specificera ett namn på ett rum. Detta kommer att visa hur anrop med parametrar sker. (Anropen till *getRoom* och *getName* hade returvärden, men inga parametrar.) Anropa funktionen *setRoom* genom att välja den från menyn. En dialog framträder och ber dig skriva in parametrarna (Figur 3.7).



Figur 3.7: Dialog för funktionsanrop med parametrar

Längst upp i dialogen visas gränssnittet för en metod som anropas (inklusive kommentarer och signatur). Nedanför finns textinmatningsfält där du kan skriva in parametrar. Signaturen överst talar om för oss att en parameter av typen String väntas. Skriv in det nya namnet på en sträng (inklusive citattecken) i textfältet och klicka på OK.

Det är allt — eftersom denna metod inte returnerar en parameter, så finns ingen resultatdialog. Anropa *getName* igen för att kontrollera att namnet verkligen har ändrats.

Lek med att skapa objekt och anropa metoder ett slag. Försök skapa en konstruktor med argument och anropa flera metoder tills du känner dig bekant med dessa operationer.

> *Summering: För att exekvera en metod, välj den från popupmenyn för objekt.*

3.3 Redigera en klass

Än så länge har vi endast sett på objektens gränssnitt. Nu är det tid att se inuti. Du kan

se implementationen av en klass genom att välja Editera Implementationen från klassoperationerna. (Kom ihåg: högerklickande på klassikonen visar klassoperationerna.) Dubbelklickande på en klassikon är en genväg för samma funktion. Editorn beskrivs inte i detalj i denna vägledning, men bör vara enkel att använda. Detaljer om editorn kommer att beskrivas separat senare. För tillfället, öppna implementationen av klassen *Staff*. Sök efter implementationen av metoden *getRoom*. Om den returnerar, som namnet antyder, är det rumsnumret för den anställde (*staff*). Låt oss byta metoden genom att lägga till prefixet "room", rum, till funktionen resultat (så att metoden returnerar, låt säga "room C.5.10" istället för bara "C.5.10"). Vi kan göra det genom att ändra raden

```
return room;
```

till

```
return "room " + room;
```

BlueJ stöder den fullständiga omodifierade Java, så det är ingenting speciellt med hur du implementerar dina klasser.

> *Summering: För att redigera källkoden till en klass, dubbelklicka dess klassikon.*

3.4 Kompilering

Efter att du satt in text (innan du gör något annat), kontrollera projektöversikten (huvudfönstret). Du kommer att se att klassikonen för *Staff* har förändrats: det är nu streckat. Dess streckade utseende markerar att klassen inte har kompilerats sedan den ändrades sist. Tillbaka till editorn.[2]

I knappraden högst upp på editorn finns några funktioner som används speciellt mycket. En av dessa är Kompilera. Denna funktion låter dig kompilera en klass direkt inifrån editorn. Klicka nu på Kompilera. Har du inte har gjort några misstag, kommer ett meddelande visa sig i informationsrutan längst ned på editorn som talar om att klassen har kompilerats. Om du har gjort misstag som lett till syntaxfel, kommer raden att markeras och ett felmeddelande visas i informationsutrymmet. (Om din kompilering fungerade första gången, försök då att lägga in ett syntaxfel — som ett uteblivet semikolon — och kompilera igen, bara för att se vad som händer.) Stäng editorn efter att du lyckats kompilera klassen.[3]

Verktygsraden för projektfönstret har också en knapp Kompilera. Denna kompileringsoperation kompilerar hela projektet.(I själva verket bestämmer den vilka klasser som behöver omkompileras och omkompilerar dessa i rätt ordning.) Prova detta genom att ändra två eller flera klasser (så att två eller flera klasser får streck i klassdiagrammet) och klicka sedan knappen Kompilera. Om ett fel upptäcks i en av de kompilerade klasserna, kommer editorn att öppnas och det ställe där felet finns visas samt felmeddelande.

Kanske har du upptäckt att objektbänken är tom igen. Objekt tas bort varje gång implementationen förändas.

> *Summering: För att kompilera en klass, klicka på knappen Kompilera i editorn. För att kompilera ett paket, klicka på knappen Kompilera i paketfönstret.*

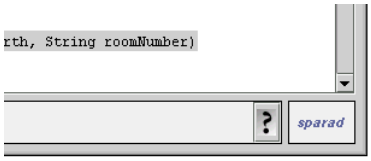
[2] Du kanske undrar varför klassens ikon var streckad när du först öppnad projektet. Det beror på att klasserna i projektet people redan var kompilerade i distributionen. Ofta är paketen med BlueJ inte kompilerade, så du kan vänta dig att se de flesta klassikoner med streck när du öppnar projekt från och med nu.

[3] Källkoden behöver inte sparas explicit. Källkoden sparas automatiskt närhelst det är lämpligt (t.ex. när editorn stängs eller före det en klass kompileras). Du kan explicit spara om du vill (det finns en funktion i editorns meny *Klass*), men det behövs endast om ditt system är instabilt, kraschar regelbundet och du är rädd att förlora ditt arbete.

3.5 Hjälp vid kompileringsfel

Mycket ofta har nybörjare svårt att förstå felmeddelanden från kompilatorn. Vi skall försöka ge lite hjälp.

Öppna editorn igen, sätt in ett fel i källkoden och kompilera. Ett felmeddelande bör visa sig i editorns informationsutrymme. Till höger om informationsutrymmet visas ett frågetecken som du kan klicka på för att få mer information om den här typen av fel (Figur 3.8).



Figur 3.8: Ett kompileringsfel och knappen för Hjälp

Vid detta tillfälle finns inte hjälptexter tillgängliga för alla fel. Några hjälptexter måste skrivas. Men det kan vara värt att försöka — många fel finns redan förklarade. De återstående kommer att skrivas och inkluderas i framtida utgåvor av BlueJ .

> *Summering: För att få hjälp med ett felmeddelande från kompilering, klicka på frågetecknet intill felmeddelandet.*

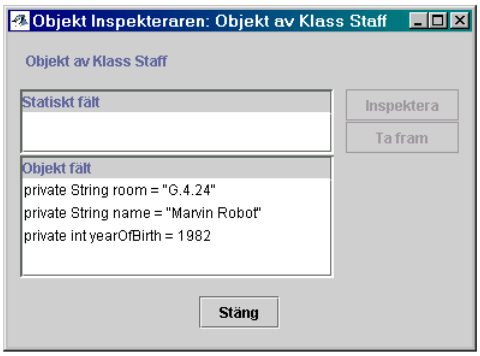
Kapitel 4

Att göra lite mer...

I detta kapitel kommer vi att gå igenom några saker du kan göra i miljön. Saker som inte är essentiella men mycket vanliga.

4.1 Inspektion

När du exekverar metoder hos ett objekt, kan du upptäckt operationen Inspektera som finns tillgänglig för objekt jämte de användardefinierade metoderna (Figur 3.5). Denna operation tillåter kontroll av tillståndet hos instansvariabler (”fält”) hos objekt. Försök skapa ett objekt med några användardefinierade variabler (exempelvis ett objekt *Staff* med konstruktorn som tar parametrar). Välj sedan Inspektera från objektmenyn. En dialog uppenbaras, visande objektfälten, deras typer och deras värden (Figur 4.1).



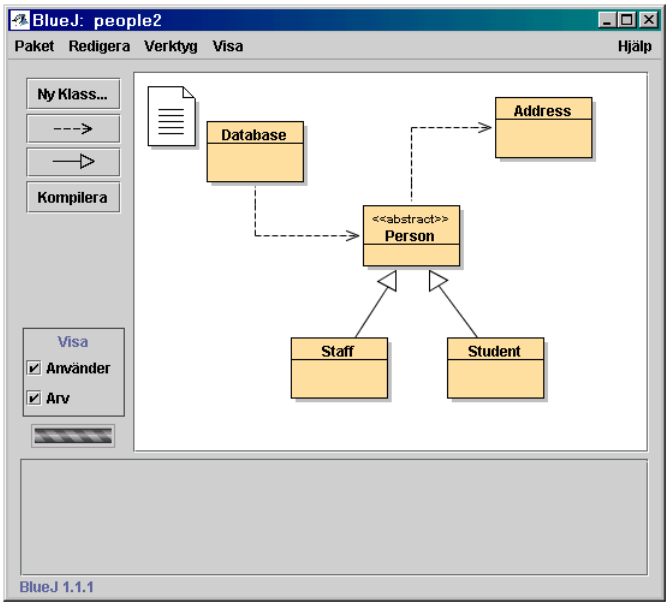
Figur 4.1: Inspektionsdialogen

Inspektion är användbart för att snabbt kontrollera hurvida en muteringsoperation (en operation som ändrar tillståndet hos objektet) exekverades korrekt. Alltså, inspektion är ett enkelt avlusningsverktyg.

I exemplet med *Staff* är alla fält av enkel typ (antingen icke-objekt typer eller strängar). Värdet hos dessa typer kan visas direkt. Du kan omedelbart se om konstruktorn har gjort de rätta tilldelningarna.

I mer komplicerade fall kan värdena hos fält vara referenser till användardefinierade

objekt. För att se ett sådant exempel använder vi ett annat projekt. Öppna projektet *people2* som också inkluderas i standarddistributionen av BlueJ. Diagrammet för *people2*, visas i Figur 4.2. Som du kan se har detta andra exempel ytterligare en klass *Address*, förutom de klasser som visades tidigare. Ett av fälten i klassen *Person* är av den användardefinierade typen *Address*.



Figur 4.2: Diagram av paketet *people2*

För nästa sak vi vill pröva ut — inspektion av objektfält — skapa ett objekt av *Staff* och anropa sedan metoden *setAddress* hos det objektet (som du finner i undermenyn till *Person*). Mata in en adress. Internt kommer *Staff* skapa ett objekt av klassen *Address* och lagra det i sitt fält address.

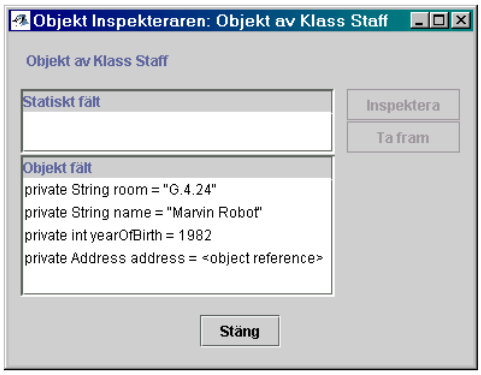
Inspektera nu objektet från *Staff*. Den resulterande inspektionsdialogen visas i Figur 4.3. Fälten inom objektet från *Staff* inkluderar nu address. Som du kan se visas dess värde som <object reference> — eftersom det är ett komplext användardefinierat objekt, kan inte dess värde visas direkt i denna lista. För att undersöka adressen närmare, välj fältet address i listan och klicka på knappen Inspektera i dialogen. (Du kan också dubbelklicka fältet *address*.) Ett annat inspektionsfönster öppnas i sin tur visande detaljerna hos objektet från *Address* (Figur 4.4).

Om det valda fältet är publikt så istället för klicka Inspektera, kan du också välja fältet *address* och klicka på knappen Ta fram. Denna operation placerar det utvalda objektet på objektbänken. Där kan du utforska det vidare genom att anropa dess metoder.

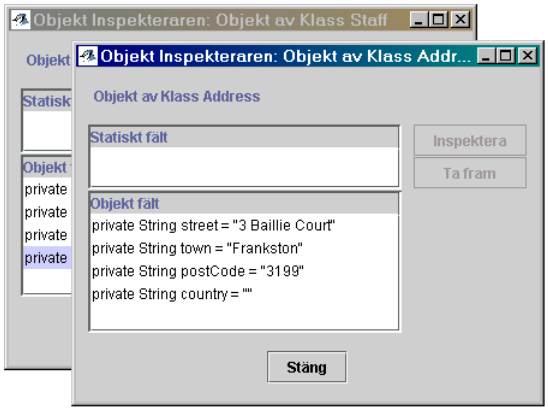
> *Summering: Objektinspektion tillåter enkel avlusning genom att visa ett objekts interna tillstånd.*

4.2 Komposition

Termen ”komposition” refererar till möjligheten att skicka objekt som parametrar till andra objekt. Låt oss försöka på ett exempel. Skapa ett objekt av klassen *Database*. (Du kommer



Figur 4.3: Inspektion med objektreferens



Figur 4.4: Inspektion av interna objekt

att se att *Database* har bara en konstruktor som inte tar några parametrar, så bildandet av objekt är enkelt.) Objektet från *Database* har möjlighet att innehålla en lista av personer. Det har operationer för att addera objekt av personer och visa alla de personer som finns inlagda. (Att kalla det för *Database*, databas, är egentligen en liten överdrift!)

Om du inte redan har ett objekt från *Staff* eller *Student* på objektbänken, skapa en av dessa också. För det följande behöver du ett objekt av *Database* samt dessutom samtidigt ett objekt från *Staff* eller *Student* på objektbänken.

Anropa nu metoden *addPerson* hos ett objekt från *Database*. Signaturen talar om att en parameter av typen *Person* väntas. (Kom ihåg: klassen *Person* är abstrakt, så det finns inga objekt som är direkt av typen *Person*. Men genom subtypning kan objekt från *Student* och *Staff* substitueras för personobjekt. Så det är tillåtet att skicka *Student* eller *Staff* där en *Person* väntas.) För att skicka ett objekt som du har på din objektbänk som parameter till det anrop du gör, kan du skriva in namnet på det i parameterfältet eller som genväg bara klicka på objektet. Detta sänder dess namn till metodanropsdialogen. Klicka OK och anropet görs. Eftersom det inte finns en returmetod för denna metod, ser vi inget omedelbart resultat. Du kan anropa metoden *listAll* i objektet från *Database* för att kontrollera att

operationen verkligen genomfördes. Operationen *listAll* skriver ut informationer om personer till standardutmatningen. Du kommer att se en terminal som automatiskt öppnas för att visa texten.

Försök igen med fler än en person i ”databasen”.

> *Summering: Ett objekt kan skickas som parameter till ett metodanrop genom att klicka på objektikonen.*

Kapitel 5

Skapa nytt projekt

Detta kapitel tar dig ut på en snabbtur hur du sätter upp nytt projekt.

5.1 Skapa ett bibliotek för projektet

För att skapa ett nytt paket, välj Paket - Nytt... från menyn. En fildialog öppnas som låter dig specificera namn och plats för ett nytt projekt. Försök det nu. Du kan välja vilket som helst namn för ditt projekt. Efter du har klickat på OK, kommer ett bibliotek skapas med det namn du valde och huvudfönstret visar det nya tomma projektet.

> *Summering: För att skapa ett projekt välj Nytt... från menyn Paket.*

5.2 Skapa klasser

Du kan nu skapa klasser genom att klicka på knappen Ny Klass på verktygsraden. Du kommer att få fylla i namnet på en klass — detta måste vara en giltig identifierare i Java.

Du kan också välja från fyra typer av klasser: abstrakt (abstract), gränssnitt (interface), appletprogram (applet) eller ”standard”. Detta val avgör vilken sorts skelett som kommer att initialt genereras för din klass. Du kan senare ändra typ av klass genom att ändra källkoden (t.ex. genom att i koden lägga till nyckelordet ”abstract”).

Efter skapandet av en klass representeras den genom en ikon i diagrammet. Olika färger identifierar olika typer av klasser, till exempel blå för normala klasser, ljusblå för abstrakta klasser, grönt för gränssnitt. När du öppnar en editor för en ny klass kommer du att märka att ett standardskelett har skapats — detta borde göra det lätt att börja. Standardkoden är syntaktiskt korrekt. Den kan kompileras men den gör inte mycket. Försök skapa en del klasser och kompilera dessa.

> *Summering: För att skapa en klass klickar du på knappen Ny Klass och specificerar klassnamnet.*

5.3 Skapa beroenden

Klassdiagrammet visar beroenden mellan klasser i form av pilar. Arvsrelationer (”extends” eller ”implements”) visas som dubbla pilar, relationen ”använder” visas som enkla pilar.

Du kan lägga till beroenden antingen grafiskt (direkt i diagrammet) eller via text i källkoden. Om du adderar en pil grafiskt kommer källkoden automatiskt uppdateras; om du lägger till i källkoden uppdateras diagrammet.

För att lägga till en pil grafiskt klickar du på den lämpliga pilknappen (dubbla pilar för ”extends” eller ”implements”, enkla pilar för ”använder”) och dra pilen från en klass till den andra.

Att lägga till pil för arv sätter in definitionerna ”extends” eller ”implements” i källkoden (beroende på om målet var en klass eller ett gränssnitt).

Att lägga till en pil för ”använder” förändrar inte omedelbart koden (om inte målet är en klass från ett annat paket. I det fallet genereras en sats ”import”, vilket vi ännu inte sett i våra exempel). Att ha en pil för använder i ett diagram pekandes på en klass som egentligen inte används i källan, kommer att generera en varning senare som talar om att ”använder” deklarerades men att klassen aldrig används.

Att lägga till pilar som text är enkelt: skriv bara in koden som du normalt skulle göra det. Så fort som klassen sparas, så uppdateras diagrammet. (Kom ihåg: stängs editorn, sparas koden automatiskt.)

> *Summering: För att skapa en pil klickar du på pilknappen och drar pilen i diagrammet, eller skriver i editorns källkodsfönster.*

5.4 Ta bort element

För att ta bort en klass från diagrammet väljer du klassen och sedan Ta Bort Klass från menyn Redigera. Du kan också välja Ta Bort från klassens popupmeny. För att ta bort en pil väljer du Radera pil från menyn och sedan väljer du pilen du vill ta bort.

> *Summering: För att ta bort en klass väljer du funktionen att Ta Bort från popupmenyn.*

> *Summering: För att ta bort en pil, välj Radera pil från menyn Redigera och klicka på pilen.*

Kapitel 6

Avlusning

Detta avsnitt introducerar de väsentligaste synpunkterna på avlusningsfunktioner i BlueJ. Vid samtal med lärare i data har vi ofta hört kommentaren att det vore trevligt att använda en avlusare under första undervisningsåret, men att det helt enkelt inte finns tid. Studenter kämpar med editorn, kompilatorn och exekveringen; det finns ingen tid över till att introducera ytterligare ett komplicerat verktyg.

Det är därför vi har beslutat att göra avlusaren så enkel som möjligt. Målet är att ha en avlusare du kan förklara på 15 minuter som studenterna kan använda framöver och de inte behöver vidare instruktioner om. Låt oss se om vi har lyckats.

Först har vi reducerat funktionaliteten av traditionella avlusare till endast tre uppgifter:

- sätta brytpunkter
- stega igenom kod
- inspektera variabler

Å andra sidan är var och en av dessa uppgifter mycket enkel. Vi kommer nu att pröva ut var och en av dom.

För att sätta igång öppnar du paketet *debugdemo*, som inkluderas i biblioteket *examples* i distributionen. Detta paket innehåller några klasser vars enda uppgift är att demonstrera avlusningsfunktionerna — därutöver är de inte speciellt begripliga.

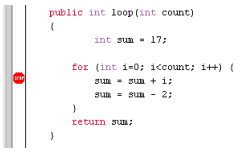
6.1 Sätta brytpunkter

Att sätta en brytpunkt låter dig avbryta körningen vid en speciell punkt i koden. När körningen avbryts kan du undersöka tillståndet hos dina objekt. Det hjälper dig ofta att förstå vad som händer i din kod.

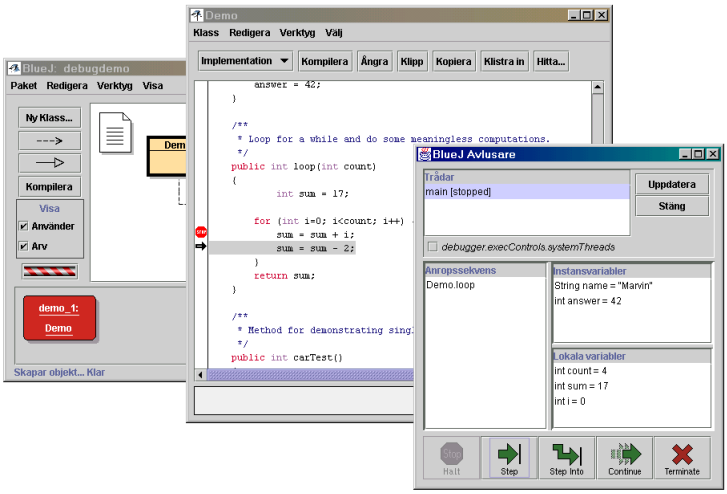
I editorn till vänster om texten finns brytpunktsutrymme (Figur 6.1). Du kan sätta en brytpunkt genom att klicka där. Ett litet stopptecken visas för att markera brytpunkten. Försök det nu. Öppna klassen *Demo*, sök efter metoden *loop* och sätt brytpunkten någonstans i loopen *for*. Stoppstecknet bör visa sig i din editor.

När kodraden med brytpunkten nås, kommer exekveringen att avbrytas. Låt oss försöka det nu.

Skapa ett objekt av klassen *Demo* och anropa metoden *loop* med en parameter, låt säga 10. Så fort som brytpunkten nås kommer ett editorfönster att öppnas, visande den



Figur 6.1: En brytpunkt



Figur 6.2: Avlusningsfönstret

nuvarande raden av kod, och avlusningsfönstret dyker upp. Det kommer ungefär att se ut som i Figur 6.2.

Markeringen i editorn visar raden som kommer att köras härefter. (Körningen stoppades före denna rad.)

> *Summering: För att sätta en brytpunkt klickar du i brytpunktsområdet till vänster om texten i editorn.*

6.2 Stega igenom kod

När vi nu har stoppat exekveringen (vilket övertygar oss om att metoden verkligen körs och denna punkt i koden verkligen nås), kan vi enkelstega genom koden och se hur körningen fortskrider. För att göra detta, klicka upprepade gånger på knappen Step i avlusningsfönstret. Du kommer att se källkodsrader i editorn förändras (markeringen förflyttas till den rad som körs). Varje gång du klickar på knappen Step, exekveras en sats och exekveringen stannar igen. Notera också att värdena hos variablerna förändras, som visas i avlusningsfönstret (t.ex. värden för *sum*). Så du kan stegvis exekvera och observera vad som händer. När du tröttnat på detta, kan du klicka på brytpunkten igen för att ta bort den, och sedan på knappen Continue i avlusningsfönstret för att åter starta exekveringen normalt.

Låt oss prova det igen med en annan metod. Sätt en brytpunkt i klass *Demo*, metoden *carTest()* i raden som lyder

```
places = myCar.seats();
```

Anropa metoden. När brytpunkten nås, kommer du precis nå en rad som innehåller ett metodanrop till metoden *seats()* i klassen *Car*. Klickar du *Step* kommer du att gå förbi raden. Låt oss pröva Step Into den här gången. Om du stegar in i ett metodanrop (step into) så kommer du att nå in i metoden och köra den rad för rad (inte stegvis bearbetning). I detta fall tas du in till *seats()* metoden i klassen *Car*. Nu kan du lyckligtvis stega dig igenom denna metod tills du når slutet och återkommer till den anropande metoden. Notera hur avlusaren visar förändringar.

Step och Step Into beter sig lika om den nuvarande raden inte innehåller ett metodanrop.

> *Summering: För att enkelstega genom din kod använder du knapparna Step och Step Into i avlusningsmiljön.*

6.3 Inspektera variabler

När du avlusar din kod är det viktigt att få insikt i sina objekt (lokala variabler och instansvariabler).

Att göra detta är trivialt — det mesta har du redan sett. Du behöver inga speciella kommandon för att inspektera variabler; instansvariabler hos det nuvarande objektet och lokala variabler hos den nuvarande metoden visas och uppdateras alltid automatiskt.

Du kan välja metoder i en anropssekvens för att se variabler hos andra aktiva objekt och metoder. Försök t.ex. sätta en brytpunkt i metoden *carTest()* igen. På vänster sida om avlusningsfönstret ser du anropssekvensen. Det visar nu

```
Car.seats
Demo.carTest
```

Detta indikerar att *Car.seats* anropades av *Demo.carTest*. Du kan välja *Demo.carTest* i denna lista för att inspektera källan och de nuvarande variablernas värden i denna metod.

Om du går förbi raden som innehåller instruktionen *new Car(...)* kan du observera att värdet hos den lokala variabeln myCar visas som <object reference>. Alla värden av objekttyp (utom *String*) visas på detta sätt. Du kan inspektera denna variabel genom att dubbelklicka den. Genom att göra det öppnas ett fönster för objektinspektering identiskt med det som tidigare beskrivits (Kapitel 4.1). Det finns ingen verklig skillnad mellan att inspektera objekt här och inspektera objekt på objektbänken.

> *Summering: Inspektera variabler är lätt - de visas automatiskt i avlusningsfönstret.*

6.4 Stanna och avsluta

Ibland när ett program körs under lång tid, funderar du kanske över om allt verkligen är riktigt. Kanske är det en infinit slinga (loop), kanske tar det bara lång tid. Nå, vi kan kontrollera. Anropa metoden *longloop()* från klassen *Demo*. Denna tar lite tid.

Nu vill vi veta vad som händer. Ta fram avlusningsfönstret om det inte redan är framme på skärmen. (Förresten, att klicka på skruven som indikerar att maskinen körs under exekveringen är en genväg för att visa avlusaren.)

Klicka nu på knappen Halt. Körningen avbryts precis som om vi hade stött på en brytpunkt. Du kan ta några steg, observera variabler och se att allt är klart — det behöver bara lite tid att bli färdigt. Du kan bara fortsätta med knappen Continue och stanna med Halt flera gånger för att se hur snabbt den räknar. Om du inte vill gå vidare (till exempel om du har upptäckt att du befinner dig i en infinit slinga) kan du bara klicka på Terminate för att avsluta hela exekveringen. Avslutningen (Terminate) bör inte användas för ofta — du kan

lämna helt välformulerade objekt i inkonsistenta tillstånd genom att abrupt avsluta maskinen, så det rekommenderas att du endast använder det som nödutgång från programmet.

> *Summering:* Halt *och* Terminate *kan användas för att stanna en körning temporärt eller permanent.*

Kapitel 7

Att skapa fristående tillämpningar

BlueJ kan skapa exekverbara jar-filer. Exekverbara jar-filer kan köras under vissa system genom att dubbelklicka på filen (t.ex. Windows), eller att utföra kommandot `java -jar <filnamn>.jar` (Unix eller DOS prompt).

Vi kommer att försöka med projektet *hello*. Öppna det (det finns i katalogen *examples*). Försäkra dig om att projektet redan är kompilerat. Välj funktionen Exportera från menyn Paket.

En dialog öppnas som låter dig välja lagringsformat. Välj "Spara som jar-fil" för att skapa en exekverbar jar-fil. För att göra jar-filen körbar, måste du också specificera en huvudsaklig klass. Denna klass måste ha en giltig *main*-metod definierad (med signaturen *public static void main(String[] args)*).

I vårt exempel är valet av huvudsaklig klass enkelt: det finns bara en klass. Välj *Hello* från popupmenyn. Om du har andra projekt, välj den klass som innehåller metoden "*main*" som du vill låta vara körbar.

Vanligtvis så vill du inte inkludera källkodsfiler tillsammans med de körbara filerna. Men du kan, om du väljer att distribuera källkodsfilerna också.

Klicka på Continue. Sedan kommer du att se en fildialog som låter dig specificera namnet på den jar-fil du vill skapa. Skriv *hello* och klicka på OK. Skapandet av exekverbara jar-filer är färdigt.

Du kan dubbelklicka jar-filen endast om tillämpningen använder ett GUI-gränssnitt. Vårt exempel använder ett textsnitt, så vi måste starta det via en textterminal. Låt oss pröva att köra jar-filen nu.

Öppna ett terminalfönster eller DOS-fönster. Gå till katalogen där du sparade din jar-fil (du bör se filen *hello.jar*). Antaget att Java är installerat korrekt på ditt system, så bör du kunna skriva

```
java −jar hello.jar
```

för att köra filen.

Summering: För att skapa en fristående tillämpning, använd Paket - Export

Kapitel 8

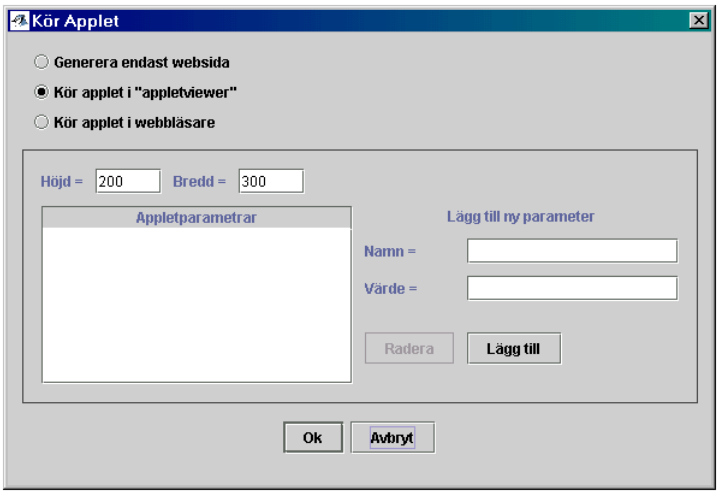
Skapa appletprogram

8.1 Köra en applet

BlueJ tillåter dig att skapa och exekvera appletprogram likaväl som tillämpningar. Vi har inkluderat några appletprogram i exempelbiblioteket i distributionen. Först vill vi försöka köra en av dessa. Öppna paketet *appletClock* från exemplen.

Du kommer att se att detta paket har bara en klass; med namnet *Clock*. Klassikonen är markerad (med bokstäverna *WWW*) som en applet. Välj kommandot Kör Applet från klassens popupmeny.

En dialog poppar upp som låter dig välja (Figur 8.1).



Figur 8.1: Dialogen "Kör Applet"

Du ser att du har möjlighet att köra appletprogrammet i en webbklient eller i en appletbetraktare (eller att generera en webbsida utan att köra det). Låt standardvärdena vara och klicka OK. Efter några sekunder kommer en appletbetraktare att dyka upp visande appletprogrammet för klockan.

Appletbetraktaren (appletviewer) är installerad tillsammans med din JDK, så den är

garanterat samma version som din Javakompilator. Generellt innebär den mindre problem än vad webbklienter gör. Din webbklient kan köra en annan version av Java och skapa problem, beroende på vilken webbklient du använder. Med de allra senaste klienterna bör det dock gå bra.

Under systemet Microsoft Windows använder BlueJ din standardklient. Under Unixsystem definieras webbklienten av inställningar i BlueJ .

> *Summering: För att köra ett appletprogram, välj Kör Applet från appletprogrammets popupmeny.*

8.2 Att skapa en applet

Efter att sett hur man kör en applet vill vi nu skapa vår egen.

Skapa en ny klass med Applet som klasstyp (du kan välja typen i dialogen Ny Klass). Kompilera och kör appletprogrammet. Det är allt! Det var inte så svårt, va?

Appletprogram (liksom andra klasser) genereras med ett standardskelett av giltig kod. För appletprogram innebär det ett enkelt program som visar två rader text. Du kan nu öppna editorn och redigera appletprogrammet för att sätta in din egen kod.

Du kommer att se appletprogrammets alla vanliga metoder finns där, var och en med kommentarer som förklarar dess syfte. Texten som visas finns i metoden *paint*.

> *Summering: För att skapa ett appletprogram, klicka på knappen Ny Klass och välj Applet som klasstyp.*

8.3 Testa appletprogram

I några situationer kan det vara användbart att skapa ett objekt av ett appletprogram på objektbänken (som i normala fall). Du kan göra det — konstruktorn visas i appletprogrammet popupmeny. Från objektbänken kan du inte köra hela appleten, men du kan anropa några metoder. Detta kan vara användbart för att testa enskilda metoder som du har skrivit som del av din implementation av appletprogrammet.

Kapitel 9

Öppna icke-BlueJ paket i BlueJ

BlueJ låter dig öppna existerande paket som har skapats utanför BlueJ. För att göra detta väljer du Paket - Öppna Icke-BlueJ... ifrån menyn. Välj katalogen som innehåller källkodsfilerna till Java och klicka sedan knappen Öppna i BlueJ. Systemet kommer att fråga efter en bekräftelse på om du vill öppna den här katalogen.

> *Summering: Icke-BlueJ paket kan öppnas med kommandot Paket: Öppna Icke-BlueJ*

9.1 Lägga till existerande klasser till ditt projekt

Ofta vill du använda en klass som du har har fått någon annanstans ifrån i ditt BlueJ-projekt. Till exempel, en lärare kan ge en Javaklass till studenterna som skall användas i ett projekt. Du kan enkelt inkludera en existerande klass till ditt projekt genom att välja Redigera - Lägg till en klass från fil från menyn. Detta låter dig importera källkoden till Java (med ett namn vars avslutning är `.java`).

När en klass importeras i ett projekt tas en kopia och sparas i den nuvarande projektkatalogen. Effekten är densamma som om du precis hade skapat en klass och skrivit all dess källkod.

Ett alternativ är att lägga till källkodsfilen för en ny klass till projektkatalogen utifrån BlueJ. Nästa gång du öppnar projektet kommer klassen att inkluderas projektdiagram.

> *Summering: Klasser kan kopieras in till ett paket utifrån genom att använda kommandot Lägg till en klass från fil i menyn Redigera.*

9.2 Anropa *main* och andra statiska metoder

Öppna projektet hello från katalogen *examples*. Den enda klassen i projektet (klassen *Hello*) definierar en standard *main*-metod.

Högerklicka på klassen och du kommer att se att klassmenyn inkluderar inte bara klassens konstruktor, men också den statiska metoden *main*. Du kan nu anropa *main* direkt från menyn (utan att först skapa ett objekt, precis som vi väntar oss från en statisk metod).

Alla statiska metoder kan anropas på detta sätt. Standardmetoden *main* väntar sig en vektor (*array*) av *String* som argument. Du kan skicka en strängvektor genom att använda Javasyntaxen för vektorkonstanter. Till exempel, kan du skicka

```
{"one", "two", "three"}
```

(inklusive klamrar) till metoden. Pröva![1]

Summering: Statiska metoder kan anropas från klassens popupmeny.

9.3 Arbeta med bibliotek

Ofta när du skriver ett Javaprogram, måste du referera till Javas standardbibliotek. Du kan öppna webbklienten som visar API-dokumentationen för JDK genom att välja Hjälp - Java Klassbibliotek från menyn (om du är uppkopplad).

JDK-dokumentationen kan också installeras och användas lokalt (nedkopplad). Detaljer förklaras i referensmanualen till BlueJ.

Summering: Javas API för standardklasserna kan visas genom att välja Hjälp - Java Klassbibliotek.

[1] I standarden för Java kan inte vektorkonstanter användas som argument till metodanrop. De kan endast användas som initialiserare. För att tillåta interaktiva anrop till standardmetoder som *main* i BlueJ, har vi tillåtit möjligheten att skicka vektorkonstanter som parametrar.

Kapitel 10

Endast summeringar

Grunderna — redigera / kompilera / exekvera

1. För att skapa ett objekt, välj en konstruktor från klassens popupmeny.

2. För att exekvera en metod, välj den från popupmenyn för objekt.

3. För att redigera källkoden till en klass, dubbelklicka dess klassikon.

4. För att kompilera en klass, klicka på knappen Kompilera i editorn. För att kompilera ett paket, klicka på knappen Kompilera i paketfönstret.

5. För att få hjälp med ett felmeddelande från kompilering, klicka på frågetecknet intill felmeddelandet.

Att göra lite mer...

1. Objektinspektion tillåter enkel avlusning genom att visa ett objekts interna tillstånd.

2. Ett objekt kan skickas som parameter till ett metodanrop genom att klicka på objektikonen.

Skapa nytt projekt

1. För att skapa ett projekt välj Nytt... från menyn Paket.

2. För att skapa en klass klickar du på knappen Ny Klass och specificerar klassnamnet.

3. För att skapa en pil klickar du på pilknappen och drar pilen i diagrammet, eller skriver i editorns källkodsfönster.

4. För att ta bort en klass väljer du funktionen att Ta Bort från popupmenyn.

5. För att ta bort en pil, välj Radera pil från menyn Redigera och klicka på pilen.

Avlusning

1. För att sätta en brytpunkt klickar du i brytpunktsområdet till vänster om texten i editorn.
2. För att enkelstega genom din kod använder du knapparna Step och Step Into i avlusningsmiljön.
3. Inspektera variabler är lätt - de visas automatiskt i avlusningsfönstret.
4. Halt och Terminate kan användas för att stanna en körning temporärt eller permanent.

Skapa fristående tillämpningar

1. För att skapa en fristående tillämpning, använd Paket - Export

Skapa appletprogram

1. För att köra ett appletprogram, välj Kör Applet från appletprogrammets popupmeny.
2. För att skapa ett appletprogram, klicka på knappen Ny Klass och välj Applet som klasstyp.

Öppna icke-BlueJ paket i BlueJ

1. Icke-BlueJ paket kan öppnas med kommandot Paket: Öppna Icke-BlueJ
2. Klasser kan kopieras in till ett paket utifrån genom att använda kommandot Lägg till en klass från fil i menyn Redigera.
3. Statiska metoder kan anropas från klassens popupmeny.
4. Javas API för standardklasserna kan visas genom att välja Hjälp - Java Klassbibliotek.