Laboration 1: De abstrakta datatyperna Stack, Kö och Lista

skriven av mbm, reviderad av dhe 2009-03-06

Syfte

Syftet med laborationen är

- att visa hur abstrakta datatyper kan skapas i Java med hjälp av klasser och interface;
- att ge förståelse för vikten av att separera mellan abstrakta datatypers gränssnitt och deras implementation;
- att ge färdighet i att använda de abstrakta datatyperna *stack* och *kö*; samt
- att illustrera hur ett gränssnitt för en *lista* kan skapas i något programmeringsspråk, till exempel Java.

Förberedelse

Ha kurslitteraturen och litteraturhandledningen för denna uppgift nära till hands när du arbetar med denna uppgift. Föreläsning 1 och 2, med tillhörande avsnitt i kurslitteraturen, är också relevanta. Det är starkt rekommenderat att du genomför de icke obligatoriska Övningarna och att du har läst igenom hela uppgiften innan du börjar arbeta.

Uppgifter

1. Skriv ett program som med hjälp av en stack testar om ett inmatat uttryck är parentesbalanserat eller inte. Programmet ska be användaren mata in en sträng, och sedan skriva ut om strängen var parentesbalanserad eller inte. Programmet ska avslutas när användaren matar in en tom sträng.
2. Skapa ett antal *testfall* som sedan används vid testning av parentesbalanseringsprogrammet. Dokumentera testresultaten med skärmdumpar ("screenshots") som läggs till som bilagor i laborationsrapporten.
3. Skriv ett menystyrt testprogram som kan användas för att testa de operationer som tillhandahålls av en given FIFO-kö. Användaren ska ha möjlighet att via olika menyval testa alla kö-operationer, till exempel att sätta in ett element (förslagsvis en textsträng) som matas in från tangentbordet. Programmet ska avslutas när användaren väljer alternativet 'Avsluta' från menyn. Om användaren försöker använda kön på ett otillåtet sätt ska ett enkelt felmeddelande skrivas ut.
4. Skapa ett antal *testfall* som sedan används vid testning av kön. Dokumentera testresultaten med skärmdumpar ("screenshots") som läggs till som bilagor i laborationsrapporten.

Gruppdiskussioner

Diskutera följande frågor/uppgifter i gruppens diskussionsforum:

1. Föreläsningsanteckningarna räknar upp några "standardoperationer" på en stack. Jämför med det givna interfacet *Stack* som beskrivs under rubriken *Genomförande*. Är det någon stackoperation som inte verkar ha någon motsvarighet i interfacet? Är det någon operation i gränssnittet för en stack som verkar överflödig i javasammanhang?
2. Jämför standardbibliotekets stack-klass (klassen *java.util.Stack*) med stack-interfacet från uppgift 1, och försök hitta likheter och skillnader.
3. När det gäller stack-ADT:n i standardbiblioteket så menar många att konstruktörerna har gjort några designmissar. Kan ni komma på några brister i designen av standardbibliotekets stack?
4. Stacken och kön i uppgift 1 och 3 använder *runtime exceptions* för att tala om för användaren att en operation har använts på ett felaktigt sätt, till exempel att en borttagningsoperation har anropats för en stack/kö som är tom. Ge exempel på ett par andra sätt för en abstrakt datatyp att signalera felaktig användning, och diskutera för- och nackdelar med de olika metoderna.
5. Hur signaleras felaktig användning av *java.util.Stack*, *java.util.Queue* respektive *java.util.List*?
6. Skriv ett Java-interface *Lista.java* för den abstrakta datatypen (osorterad) *Lista*. Interfacet behöver inte innehålla alla tänkbara operationer på en osorterad lista, men åtminstone de viktigaste standardoperationerna (skapa lista, förstöra lista, kontrollera om listan är tom/full, antal element) och några varianter av insättnings- och borttagningsoperationerna.

Utse någon i gruppen som sammanfattar diskussionen om punkterna 1-5, och kom överens om ett förslag på interface i sista uppgiften. Posta gruppens gemensamma lösning i kursens diskussionsforum vid den tid som anges i Blackboard.

Genomförande

Börja med att skapa en katalog för laborationen på ditt konto eller din dator, och spara ner följande jar-filer till den nyskapade katalogen:

1. stack.jar
2. queue.jar
3. lab1classes.jar

Om de första två filerna inte packas upp automatiskt, gör det med hjälp av något packprogram eller med hjälp av kommandot jar: *jar xvf stack.jar*

När du packar upp *stack.jar* så kommer katalogen 'stack', som innehåller några underkataloger och ett antal filer, att skapas i den aktuella katalogen. På samma sätt kommer katalogen 'queue' att skapas när du packar upp *queue.jar*.

Den tredje jar-filen innehåller färdigkompilerade class-filer som du behöver för att kunna kompilera och testköra programmen i laborationen. Du kan förslagsvis lägga jar-filen i den katalog som du skapade för laborationen. Den behöver inte packas upp.

Uppgift 1. Testprogram för en stack: Balanserade parenteser

Börja med att studera de metoder som specificeras i interfacet *Stack* (filen Stack.java).

Interfacet *Stack* implementeras av den färdigkompilerade klassen *ListStack*. Klassen tillhör paketet *se.hig.ad1.lab1* och källkoden finns i katalogen 'stack/src'. Dokmentation av interfacet finns i katalogen 'stack/javadoc'.

Börja nu implementera testprogrammet, som alltså ska undersöka om stränguttryck som matas in av användaren innehåller balanserade parentesuttryck eller ej. Följ kravspecifikationen i uppgiftsbeskrivningen!

Exempel på balanserade parentesuttryck: (), (()), ()(), ((())())

Exempel på obalanserade parentesuttryck:)(, (((), ()), (()(

Utgå gärna från den skelettkod som finns i filen *CheckBalance.java*; se jar-filen. Skriv den statiska metoden `isBalanced`, som tar ett parentesuttryck (en textsträng) som inparameter, och som returnerar *true* om uttrycket är parentesbalanserat, annars *false*. Anropa metoden från huvudprogrammet där du läser in ett parentesuttryck från tangentbordet.

Testprogrammet får gärna ha ett grafiskt användargränssnitt, men ett enkelt textbaserat gränssnitt räcker. Det går bra att använda Console-klassen från övning 1.

Algoritm för balanserade parenteser

Algoritmens grundidé beskrivs i avsnitt 6.6.2 i kurslitteraturen. Här är samma algoritm i en mer detaljerad pseudokod:

```
Skapa en tom teckenstack
Antag att uttrycket är balanserat
Så länge uttrycket är balanserat
och det fortfarande finns tecken i strängen
    Hämta nästa tecken från strängen
    Om nästa tecken är '('
         Placera '(' på stacken
    Annars
        Om nästa tecken är en ')'
            Om stacken ej är tom
                Plocka bort toppen av stacken
            Annars
                Markera att uttrycket ej är balanserat
                // Vi har hittat en högerparentes som saknar maka
Om uttrycket är balanserat och stacken ej tom
    Markera att uttrycket ej är balanserat
    // Vi har kvar vänsterparenteser som saknar makar
Förstör stacken
```

Tips

1. För att kunna kompilera programmen behöver du tala om för kompilatorn att de filer som finns i *lab1classes.jar* ska ingå i projektets classpath. Om du arbetar i terminalfönster görs det genom att ange flaggan *-classpath* (följt av filens namn, inklusive sökvägen till filen) till kompilatorn. Om du arbetar med någon IDE lägger du till filen i projektets sökvägar. Hur det går till skiljer sig åt mellan olika utvecklingsmiljöer.

2. För att komma åt ett tecken på en viss position i strängen, använd metoden *charAt*. Se avsnitt 2.3.4 i kurslitteraturen.
3. De parenteser som pushas på stacken är av typen `char`, men elementtypen i stacken är `Object`. Hur löser vi detta problem?

   Lösningen är att först "slå in" tecknet i en så kallad *wrapper-klass*. För just tecken finns wrapper-klassen `Character`. Följande mönster kan användas för att stoppa in ett tecken i stacken:

   ```
   char ettTecken;
   ...
   ettTecken = parentesuttrycket.charAt(i); // Plocka ut det i:te
   tecknet
   stacken.push(new Character(ettTecken));  // "Slå in" tecknet och
   pusha på stacken
   ```

   I ovanstående exempel kommer Java vid insättningen i stacken att göra en *implicit typkonvertering* från klassen `Character` till klassen `Object`, eftersom inparametertypen för metoden *push* är just Object. Klassen `Object` är den basklass som alla andra klasser i Java ärver egenskaper från. Det innebär också att alla objekt i Java också är `Object`-objekt. Läs mer om detta i avsnitt 4.5 - 4.5.1 i kurslitteraturen.

   Varför måste vi göra på detta sätt? Svaret är att en `char` är en av de primitiva datatyperna, och de primitiva datatyperna är inga objekt. Alltså ärver de inte egenskaper från klassen Object, till skillnad från vår "wrapper" (Character) som gör det. Alltså kan Character-objekt pushas på stacken, men däremot inte char-variabler. Läs mer om wrapper-klasser i kurslitteraturen, avsnitt 4.6.2. Notera att exemplet i figur 4.24 använder ungefär samma mönster som beskrivs ovan.

   Ett alternativ till detta mönster är att använda *generics*, som introducerades i Java 2 version 1.5. Se litteraturhandledningen!

Uppgift 2. Tester

Skapa ett antal "testfall" för parentebalanseringsprogrammet och testa programmet utgående från dessa testfall. Dokumentera utfallet av testerna genom att lägga skärmutskrifterna som en bilaga till rapporten. Beskriv testfallen i rapportens resultatdel och hänvisa till bilagan med testresultaten, så att det tydligt framgår av rapporten att stacken fungerar. Exempel på lämpliga testfall:

- 1.1 Kontroll av uttrycket ()
- 1.2 Kontroll av uttrycket)(
- 1.3 Kontroll av uttrycket (()
- ...

Uppgift 3. Testprogram för en kö

Försök att med papper och penna skapa ett Java-interface för en *kö* (FIFO-kö). Specificera vilka metoder som ingår i interfacet, deras eventuella inparamterar och deras returtyper. Diskutera gärna med din basgrupp om du kör fast eller om något är oklart. När du är klar, jämför ditt eget förslag med interfacet *Queue.java*; se jar-filen.

Interfacet *Queue* implementeras av den färdigkompilerade klassen *ListQueue* i paketet *se.hig.ad1.lab1*, och källkoden finns i katalogen 'lab1/queue/src'. Dokmentation av interfacet finns i katalogen 'lab1/docs'. Observera att vissa metoder kastar en "runtime exception" (klassen *QueueIsEmptyException*) om användaren försöker att komma åt ett element i en tom lista.

Skapa nu det menystyrda testprogrammet för kön, enligt den givna kravspecifikationen. Utgå gärna från skelettkoden i filen *TestQueue.java*; se jar-filen.

Tips

Ignorera till en början eventuella undantag som kastas av någon av metoderna i klassen ListQueue. När du har fått programmet att fungera, gå tillbaka till de menyval där du anropar metoder som kan kasta undantag. Se till att undantagen fångas upp, och plocka fram undantagsobjektets meddelandetext. Skriv sedan ut den som ett felmeddelande i terminalfönstret.

Uppgift 4. Tester

Testa FIFO-kön på ungefär samma sätt som parentesbalanseringsprogrammet testades i uppgift 2. Här är några exempel på lämpliga testfall:

- 1.1 Insättning av element i tom kö
- 1.2 Insättning av element i en icke-tom kö
- 2.1 Försök att ta bort element från en tom kö
- 3.1 Borttagning av element från icke-tom kö...

Fråga 2-3.

I Javas standardbibliotek finns klassen *java.util.Stack*, som är en implementation av en stack-ADT. Vi kommer inte använda oss av standardbibliotekets stack i programmeringsuppgifterna, men vi bör känna till att den finns.

Fråga 5.

Använd testprogrammen från övning 2 i kombination med studium av API-dokumentationen för de aktuella klasserna.

Fråga 6.

Varje gruppmedlem skapar ett eget interface i filen *Lista.java*. Låt förslagsvis elementtypen för listan vara *Object*, så att vi får en generisk lista* som kan innehålla vilken typ av element som helst. Följ exemplet i interfacen *Stack.java* och *Queue.java*. Observera att du inte behöver skapa någon klass som implementerar interfacet. Du kan alltså inte testköra ditt listinterface, men se till att kompilera interfacet och rätta eventuella kompileringsfel. Jämför ditt förslag med dina gruppmedlemmars förslag. Diskutera fram ett "vinnande" förslag eller jämka ihop era förslag till ett slutligt förslag från gruppen.

* Alternativt kan du använda dig av *Generics* för att skapa listor vars elementtyp bestäms när listan skapas.

Redovisning

Programmeringsuppgifterna och rapportskrivning utförs *gruppvis.* Följande är en checklista över alla delar som ingår i redovisningen:

- En skriftlig laborationsrapport som innehåller svar på de frågor som ställs i uppgiften, och i övrigt utgår från de typografiska mallar som finns tillgängliga på Blackboard. Glöm inte att ange författarens/författarnas namn, personnummer och e-postadress på rapportens framsida. Lämna inte in Word-filer, på grund av risken för spridning av makrovirus. Konvertera istället till PDF-format. Instruktioner för hur du gör det finns på Blackboard.
- Dokumentation av testresultaten i form av skärmdumpar.
- Alla källkodsfiler som hör till uppgiften, även de som du inte själv har skrivit/modifierat.
- javadoc-genererad HTML-dokumentation av de klasser du själv har skrivit/modifierat.

Packa ihop laborationsrapporten, skärmdumpar, källkoden och javadoc-dokumentationen i en jar-fil eller zip-fil, och ladda upp filen till gruppens ”File Exchange” i Blackboard.

Gruppdiskussionen genomförs i respektive basgrupp. En sammanfattning av gruppens svar på frågorna läggs till som en bilaga i gruppens gemensamma laborationsrapport. Glöm inte att ange gruppmedlemmarnas namn på första sidan av rapporten. Aktivt deltagande i gruppdiskussionen är en förutsättning för att få godkänt på hela laborationsuppgiften.

Lycka till! :