EDAA45 Programmering, grundkurs

Läsvecka 14: Tentaträning

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2016

14 Tentaträning

- Tentatips
- Avgränsning
- Tips vid val av lösningar
- Genomgång av extenta

Tentatips

Före tentan:

1. Repetera övningar och labbar i kompendiet.
2. Läs igenom föreläsningsanteckningar.
3. Studera **snabbref** **mycket noga** så att du vet vad som är givet och var det står, så att du kan hitta det du behöver snabbt.
4. Skapa och **memorera** en personlig **checklista** med programmeringsfel du brukar göra, som även inkluderar småfel, så som glömda parenteser och semikolon, och annat som en kompilator/IDE normalt hittar.
5. Tänk igenom hur du ska disponera dina 5 timmar på tentan.
6. Gör den minst en extenta som om det vore **skarpt läge**:
    1. Avsätt 5 ostörda timmar (stäng av telefon, dator etc).
    2. Inga hjälpmedel. Bara snabbref.
    3. Förbered dryck och tilltugg.

På tentan:

1. Läs igenom **hela** tentan först.
   Varför? Förstå helheten. Delarna hänger ihop.
2. Notera och begrunda specifika begrepp och definitioner.
   Varför? Begreppen är avgörande för förståelsen av uppgiften.
3. Notera förenklingar, antaganden och specialfall.
   Varför? Uppgiften blir mkt enklare om du inte behöver hantera dessa.
4. **Fråga** tentamensansvarig om du inte förstår uppgiften – speciellt om det finns misstänkta felaktigheter eller förmodat oavsiktliga oklarheter.
   Varför? Det är inte lätt att konstruera en ”perfekt” tenta.
   Du får fråga vad du vill, men det är inte säkert du får svar :)
5. Läs specifikationskommentarerna och metodsignaturerna i alla givna klass-specifikationer **mycket noga**.
   Varför? Det är ett vanligt misstag att förbise de ledtrådar som ges där.
6. Återskapa din memorerade personliga checklista för vanliga fel som du brukar göra och avsätt tid till att gå igenom den på tentan. Varje fix plockar poäng!
7. Lämna in ett försök även om du vet att lösningen inte är fullständig. Det gäller att ”plocka poäng” på så mycket som möjligt. En dålig lösning kan ändå ge poäng.
8. Om du har svårigheter kan det bli kamp mot klockan. Försök hålla huvudet kallt och prioritera utifrån var du kan plocka flest poäng. Ge inte upp! Ta en kort äta-dricka-paus för att få mer energi!

Planeringstips

Exempel på saker som du kan lägga in tid för i din julpluggkalender:

1. Välja ut övningar att repetera
2. Repetera övning X, Y, Z, ... Både läsa och skriva kod. Fundera på typ och värde.
3. Välja ut labbar att repetera
4. Repetera labb X, Y, Z, ... Lär dig ”trick” och ”mönster”.
5. Träna på att skriva program med papper och penna
6. Göra checklista med vanliga fel
7. Läsa igenom extentor i Java
8. Välja ut minst en Java-extenta att göra som i skarpt läge i Scala
9. Gör Java-extentor X, Y, Z, ... implementera (delar) i Scala
10. Gör utvalda delar av extenta X, Y, Z, ... i Java

Avgränsning

Tentans struktur

- Del A 20%:
  Läsa uttryck där du ska **ange typ och värde**
  - Du kommer att behöva skriva ner delsteg och variablers värden (minnet)
  - Testar förståelse av variabler, uttryck, samlingar, algoritmer, arv, etc.
  - Uppdaterad (mildare) regel om ”rättningströskel”:
    Ur senaste compendium.pdf kapitel 0.8: *Om du på del A erhåller färre poäng än vad som krävs för att nå upp till en bestämd ”rättningströskel”, kan din tentamen komma att underkännas utan att del B bedöms.*
  - Liknar kompendiets övningar; rimlig att lösa och dubbelkolla på 1h
- Del B 80%:
  Skriva kod som uppfyller **krav och designspecifikation**
  - Testar att du själv kan skapa kod med delar som samverkar
  - Testar förmåga att gå från indata-utdata till algoritm
    givet: ledtrådar, design, ev. skiss på lösning, ev. pseudokod etc.
  - Liknar kompendiets labbar; rimlig att lösa och dubbelkolla på 4h

Vad kommer på tentan? (1 av 3)

Allmänt:

- Begrepp som är ”fördjupning” krävs ej på tentan (men ökar förståelse)
- Ok om du väljer en enklare lösning med basala begrepp som fungerar bra, i stället för en kortare/elegantare/mer avancerad lösning
- Dessa moduler ingår ej på tentan: ”Trådar, webb”, ”Design, api”

Modul	*Ingår t.ex.*	*Avgränsning* (ej krav; ok anv. om lämpl.)
Introduktion	uttryck, aritmetik, slumptal, strängar, typer, Unit skillnad mellan heltal & flyttal variabler, for, while, if	kan ha nytta av deMorgan men ej krav skriva egna s`"$x"` (men kunna läsa) Float, Byte, Short hex-literaler, backticks
Kodstrukturer	iterering, SWAP, SUM, MIN/MAX loopar, Range, sats vs uttryck namn, synlighet, skuggning	import, paketnamn ok att välja vilken loop du tycker passar scaladoc, javadoc, jar
Funtioner, objekt	definiera, anropa, parameter returtyp, namnarop, defaultarg punktnotation, objekt vs static map/foreach med egen funktion anonyma funktioner (lambda)	skapa egen kontrollstruktur stegad funktion, rekursion lazy val

Vad kommer på tentan? (2 av 3)

Modul	*Ingår t.ex.*	*Avgränsning* (ej krav; ok anv. om lämpl.)
Datastrukt.	attribut, medlem, metod, klass tupler, Vector, Set, Map Source.fromFile	isInstanceOf (anv. match istället) List (oftast Vector istället) java.nio.file
Sekvensalg.	skapa ny samling från befintlig registrering, Scanner, ArrayBuffer uppdatera Array, ArrayBuffer, Vector slumptalsfrö, scala.util.Random	StringBuilder
Klasser	new, this, synlighet inkapsling, accessregler, private klassparameter, fabriksmetod class vs case class referenslikhet vs innehållslikhet föränderlig vs oföränderlig klass	null private[this]
Arv	bastyp, subtyp, trait, extends överskuggning, Any, AnyVal, AnyRef, Object accessregler vid arv, protected abstract class, case object	 inmixning, Null, Nothing final

Vad kommer på tentan? (3 av 3)

Modul	*Ingår t.ex.*	*Avgränsning* (ej krav; ok anv. om lämpl.)
Mönster	match, Option, Try	try catch, unapply
	flatten, sealed	flatMap, partiella funktioner
	enkel equals utan arv	hashcode, fullständig equals
	wildcard-mönster	variabelbindn. i mönster, sekvensmönster
Matriser, typparametrar	indexering i nästlade strukturer	
	nästlad for-sats	
	matriser i Java med array	
	använda generiska strukturer	skapa generiska strukturer
Sök, sortera	linjärsökning, binärsökning	algoritmisk komplexitet
	compareTo, strängjämförelse	Ordering, Ordered
	insättningssortering	räcker kunna en valfri sortering
Scala/Java	översätta enkel Java/Scala	try catch i Java
	implemenetera Java-klass	arv i Java med super vid konstr.
	grundläggande syntaxskillnader	
	ArrayList vs ArrayBuffer	java.util.{List, Set}
	OBS! Java-övningar finns även här och där i andra moduler	

Tips vid val av lösningar

Tips om val av klass/trait

Ofta ger tentan en specifik design, men du kan ha stor nytta av egna abstraktioner, speciellt **lokala funktioner** för att göra enklare dellösningar!

Tips om val av klass/trait

Ofta ger tentan en specifik design, men du kan ha stor nytta av egna abstraktioner, speciellt **lokala funktioner** för att göra enklare dellösningar!

Om du skulle behöva samla både attribut och metoder utöver givan specifikationer: Singelobjekt, case-klass, klass, trait eller abstrakt klass?

- Använd `object` om du behöver samla metoder (och variabler) i en modul som bara finns i en upplaga. Du får lokal namnrymd och punktnotation på köpet.
- Använd en `case class` om du har **oföränderlig data**. Du får då även innehållslikhet, möjlighet till mönstermatchning, etc. på köpet!
- Behöver du **föränderligt tillstånd** använd en vanlig `class`.
  Det normala är att tillståndet (alla attribut) är `private` eller `protected` och att all uppdatering och avläsning av tillståndet sker indirekt genom metoder (getters/setters/...).
- Behöver du en abstrakt bastyp utan konstruktorparametrar använd en `trait`. (Du får inmixningsmöjlighet med `with` på köpet. Inmixning kommer ej på tenta.)
- Behöver du en abstrakt bastyp med konstruktorparametrar använd en `abstract class`. (Går dock ej att använda vid inmixning med `with`.)

Tips om hur man läser en specifikation

När du läser en specifikation av en klass, en trait, eller ett singelobjekt:

- Tänk igenom vilket ansvar olika delar av koden har
- Vad håller klassen reda på?
  $\rightarrow$ Ledtrådar till attribut
- Vad ska klassen göra/räkna ut?
  $\rightarrow$ Ledtrådar till metoder och deras algoritm
- Vilka andra klasser har nytta av denna metod?
  $\rightarrow$ Ledtrådar till hur klasserna samverkar för att lösa uppgiften

Rita gärna en bild med ett specifikt exempel på vilken data som olika instanser håller reda på och fundera på hur data skapas/beräknas/förändras

Tips om val av samling

Generellt: Det är ofta enklare med oföränderliga samlingar med oföränderliga element och skapa nya samlingar vid förändring. Men ibland blir det enklare om man har föränderliga samlingar.

- Behöver du hantera värden `x:Typ` med **heltalsindex**?
  - Om du klarar dig utan förändring av innehållet:
    `val xs: Vector[Typ]`
  - Om du behöver ändra innehåll men **inte** antal element:
    `val xs: Array[Typ]`
  - Om du behöver ändra innehåll **och** antal element:
    `var xs: Vector[Typ]` (se metoden `patch`) eller
    `val xs: ArrayBuffer[Typ]` (har metoden `insert`)
- Behöver du hantera värden `x:Typ` som är unika?
  - Oföränderlig: `val xs: Set[Typ]`
  - Förändringsbar: `val xs: scala.collection.mutable.Set[Typ]`
- Behöver du leta upp värden `x:Typ` utifrån en nyckel av typen String?
  - Oföränderlig: `val xs: Map[String, Typ]`
  - Förändringsbar:
    `val xs: scala.collection.mutable.Map[String, Typ]`

Tillåtna uppdateringar i din QuickRef

Du får med egen penna göra dessa fixar i din QuickRef:

- Grundtypernas implementation, sid 4:
  - omfång för Int ska ha exponent 31 (inte 15),
  - omfång för Long ska ha exponent 63 (inte 15).
- Saknade samlingsmetoder:
  - Under rubriken "Methods in trait Map[K, V]" saknas metoderna keySet och mapValues.
  - Saknade metoderna för mutable.ArrayBuffer[T]: `update` `insert` `remove` `append` `prepend`, etc. lägg till beskrivning på lediga platsen på sista sidan se vidare commit a5e29d000062a

ArrayBuffer

Viktigast att känna till: update, insert, remove, append

`xs(i) = x`		Replace element at index i with x.
`xs.update(i, x)`		Return type Unit.
`xs.insert(i, x)`		Insert x at index `i`. Remove element at i.
`xs.remove(i)`		Return type Unit.
`xs.append(x)`	`xs += x`	Insert x at end. Return type Unit.
`xs.prepend(x)`	`x +=: xs`	Insert x in front. Return type Unit.
`xs -= x`		Remove first occurance of x (if exists). Returns xs itself.
`xs ++= ys`		Appends all elements in ys to xs and returns xs itself.

Genomgång av extenta

Extenta 2016-08-24 TimePlanner

`http://cs.lth.se/pgk/examination/`

TimePlanner:

- tentamen 160824
- lösningsförslag Java
- översättning av lösning till Scala