OOPD
Eksamen Januar 2007
"Galapagos"

Marie Lund Christophersen
René Hansen
Kasper Henriksen
Anders Bjerg Pedersen

19. januar 2007

Indhold

0.0.1 Formalia . . . 2
0.0.2 Formål . . . 2

1 Programdesign **3**
1.1 Modellen . . . 3
1.1.1 Finker og strategier . . . 3
1.1.2 Biotopen . . . 3
1.2 MVC-mønster og GUI-design . . . 5
1.2.1 Klassen Galapagos . . . 5
1.2.2 Klassen Biotope . . . 6
1.2.3 Klassen Controller . . . 6
1.2.4 Klassen BiotopeViewer . . . 6
1.2.5 Klassen PieChart . . . 6
1.2.6 Klassen BiotopeLogger . . . 7
1.2.7 Klassen InitializerDialog . . . 8
1.3 JUnit-tests . . . 8

2 Konklusion **9**

0.0.1 Formalia

Projektet er forfattet af:

Kasper Henriksen
Eksamensnummer: 4
Tårnhusvej 5
4690 Haslev

Anders Bjerg Pedersen
Eksamensnummer: 48
Ørholmgade 2, 1.th.
2200 København N

René Hansen
Eksamensnummer: 8
Spindestræde 11, st.th.
2635 Ishøj

Marie Lund Christophersen
Eksamensnummer: 45
Frederiksborgvej 32, 1.
2400 Kbh. NV

0.0.2 Formål

Formålet med nærværende projekt er ved brug af teknikker fra objektorienteret programmering og design at designe, implementere og afprøve en model af en biologisk biotop, hvori der er udsat finkebestande. Disse finkebestande skal kunne tildeles forskellige livsstrategier til at bestemme deres opførsel over for andre finker, og det færdige program skal virke som et visualiseringsværktøj til at overvåge de forskellige bestande løbende over et givent tidsrum. Desuden skal det overvejes, hvordan man kan udvide modellen til at have flere dyrearter, forskellige områdetyper og bevægelse af dyrene.

1 Programdesign

Programmet består af to dele, selve modellen og en brugerflade, der formidler brugerens ønsker til biotopen og dens kørsel til modellen. Følgende afsnit beskriver design og implementering af selve modellen, altså vores biotop og de metoder, hvorpå vi holder styr på indholdet af biotopen og karakteriserer de forskellige former for indhold, biotopen kan rumme.

1.1 Modellen

Vi har valgt at opdele modellen i to hoveddele: selve biotopen, og finkerne og deres strategier. Dette er gjort for senere at lette en eventuel udvidelse af modellens muligheder (andre dyrearter og andet). I det følgende vil vi kort beskrive de to hovedkomponenter, både mht. implementering og fordele/ulemper ved de valg, der er foretaget. Se desuden Figur 4 på side 10.

1.1.1 Finker og strategier

Overordnet har vi generaliseret finker til en `Animal`-interface, så man senere kan tilføje andre dyretyper. Brugerfladen `Finch` udvider `Animal`-interfacet, og den fælles kode for alle de forskellige finker er implementeret i `AbstractFinch`. I de forskellige finke-klasser (f.eks. `CheaterFinch`), der udvider `AbstractFinch`, implementeres de for finkerne forskellige konstruktører og metoden `giveBirth`, der (som navnet antyder) bruges, når finkerne føder nye unger af samme slags.
Når en ny finke fødes (hvilket i vores implementation sker både i starten, når biotopen fyldes, og løbende i simulationen), tildeles den et strategi-objekt, som er en instans, der implementerer `Behaviour`, f.eks. `Cheater.java`, som implementerer metoderne `help` og `remember`. `help` afgør, om finken ønsker at hjælpe den finke, den møder, og `remember` bruges af de finker, der har brug for at huske eventuelle "onde" eller "gode" finker til efterfølgende møder.

Den umiddelbare fordel ved at bruge separate strategi-objekter er muligheden for at skifte strategi undervejs uden alt for megen besvær. Skal en finke skifte strategi, tildeles den blot et nyt strategi-objekt, som den så efterfølgende handler efter.
Ulempen ved denne implementation er, at antallet af klasser, der skal vedligeholdes, er forholdsvis stort.

Vi vil kort nævne, at vi har kreeret to typer designer-finches: en Grumpy Old Finch og en Trauma Finch. Grumpy Old Finch hjælper med mindre og mindre sandsynlighed, jo ældre den bliver. Vores Trauma Finch hjælper alle, indtil den selv bliver afvist for første gang. Herefter er den traumatiseret og kan aldrig hjælpe andre finker mere. Den er altså en metafinke, der starter med at have samaritanerstrategi og derefter skifter til cheaterstrategi.

1.1.2 Biotopen

Til at modellere biotopen har vi valgt en enkelt overordnet klasse `Biotope`, der indeholder et todimensionelt array. Hver plads i arrayet indeholder et objekt af typen `Square`, som igen indeholder et objekt af typen `Animal`. Dette er gjort af udvidelseshensyn, da man

senere vil kunne lave `Square` til at indeholde et objekt af typen område. Et område kunne være ørken, jungle,eller vådområde, der har hver deres egenskaber og fordele/ulemper for forskellige dyrearter. `Biotope` sørger ligeledes for at vedligeholde indholdet af vores biotop (vores array) jævnfør de 4 trin i en "runde"i simulationen. Sidst men ikke mindst varetager den diverse statistikker for biotopens forandringer efter hver runde.

En anden stor fordel ved vores implemantation er, at den er uafhængig af, om vi arbejder med finker eller en hvilken som helst andet dyreart. Der er altså ikke hardcoded noget til brug for finker i `Biotope`. Derfor er det i princippet ikke nødvendigt at ændre i `Biotope` for at tilføje dyrearter.

De 4 trin i en runde er i vores implementation inddelt i to faser hvor vi gennemløber biotopens felter:

Først gennemløbes vores array med metoden `birthFase`, og hvor der står finker med status lig *ALIVE*, kan disse nu føde på et tilfældigt frit nabofelt, hvis de rammer fødesandsynligheden. De nyfødte finker får sat status lig *BABY* og kan dermed ikke selv føde i samme runde. Samtidig opdateres fødestatistikken og populationsstørrelserne.
I andet gennemløb udføres de resterende 3 trin med metoden `meetPayAndKillFases`: der indgås møder finkerne imellem, og finkerne ældes, rundeprisen trækkes, og de finker, der er døde, får sat status lig *DEAD_FROM_TICKS* eller status lig *DEAD_FROM_AGE*. Samtidig opdateres dødstallene og populationsstørrelserne.

Disse kun to gennemløb er med til at effektivisere hver runde, der bliver kørt i biotopen. Dermed ender vores model med at få tidskompleksitet $O(n)$, dvs. den er lineær i størrelsen af biotopen og også lineær i antallet af runder, der køres.

Vi har overvejet om man ikke kunne få dette tal ned, så man kun skulle gennemløbe arrayet én gang. Dette kan desværre ikke lade sig gøre, idet en nyudklækket finke stadig skal have lov til et møde i samme runde. Så hvis den bliver udklækket på et felt til venstre og oven for den finke, der udklækker den, vil den ikke møde nogen finke den selvsamme runde, idet arrayet gennemløbes fra venstre mod højre, og fra oven og nedefter.

Når biotopen konstrueres, laver den alle finkerne, ved at kalde `giveBirth`-metoden et antal gange hos en finke af hver slags. Finkerne instantieres altså ved et fabriksmønster. Grunden til, at dette er en fordel, er, at man kan foretage et metodekald og få den rigtige type finke sat ind i biotopen uden at behøve at kende typen af finken. Med andre ord undgår man ting som `instanceof`.

Man kunne overveje at udvide modellen, så finkerne kan bevæge sig. Dette ville ikke være vanskeligt, man skulle blot gennemløbe arrayet en gang til og gøre det muligt for finkerne at gå til frie nabopladser og holde styr på hvem, der allerede har rykket sig. Vi har dog ikke implementeret dette, da der allerede nu er stor bevægelse i biotopen, når der bliver født nye finker.

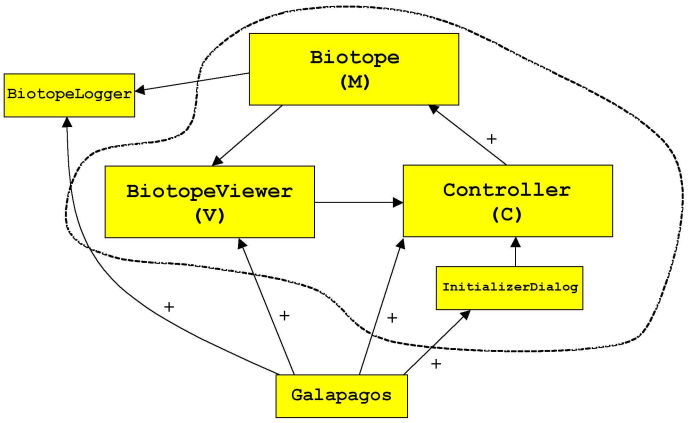
1.2 MVC-mønster og GUI-design

Vi benytter et MVC-mønster til at få modellen og den grafiske brugerflade til at hænge sammen. Klassen `Biotope` står for al kontakt ud af modellen, dvs. den udvider klassen `Observable`. `BiotopeViewer` er den grafiske brugerflade og implementerer `Observer`. `Controllor` er kontrolløren, der implementerer `ActionListener`. Det smarte ved denne metode er, at modellen ikke ved noget om de klasser, der observerer den. Dermed er den helt uafhængig af den grafiske brugerflade, og denne kan derfor "nemt" ændres, hvis man har behov derfor.

For at man nemt skal kunne specificere parametrene på modellen, har vi valgt at lave en dialogboks, der popper op når man starter en ny biotop. Man kan her også vælge hvilke finketyper man gerne vil have med i sin biotop. Det er klassen `InitialzerDialog`, der udvider `JDialog`, som laver en sådan dialogboks. `Controller`, der i forvejen lytter til `BiotopeViewer`, får kaldt sin `actionPerformed`-metode af dialogboksen, og kan på den måde instantiere en biotop med de specificerede værdier.

Loggen har vi valgt også at lave som observatør på biotopen, for at modellen heller ikke skal afhænge af den. Den er altså lavet efter samme princip som et MVC-mønster, bortset fra at vi ikke har behov for en kontrollør, da der ikke er interaktion med en bruger, og dermed kan loggen ikke ændre noget.

GUI'en er har vi valgt at hardcode specifikt til finker, da det ville involvere for stort et arbejde at skalere denne til flere dyrearter og områdetyper.



Figur 1: Oversigt over MVC-pattern og GUI-design

1.2.1 Klassen Galapagos

Denne klasse har `main`-metoden, som starter programmet. Desuden opretter den en kontrollør og de to observatører, `BiotopeViewer` og `BiotopeLogger`, og sørger for, at de kender kontrolløren.

1.2.2 Klassen Biotope

`Biotope` formidler al kontakt fra modellen til brugerfladen. Den implementerer `Observable`, og den gør opmærksom på ændringer til sine observatører i to metoder. Den ene metode er `init`. Der sker ikke andet i denne metode end at observatørerne bliver gjort opmærksomme på, at biotopen er oprettet, så de kan hente initialværdierne. Den anden metode er `advanceTime`, der sørger for at køre en hel runde. Observatørerne bliver dermed opdateret efter hver runde.

Klassen har nogle offentlige metoder til at hente statistik ud med, og én der giver hvilket dyr, der står på en given plads i arrayet. Med disse metoder kan `BiotopeViewer` konstruere billedet, der viser, hvor finkerne står, og loggen kan udskrive sin statistik.

1.2.3 Klassen Controller

`Controller` kontrollerer biotopen og lytter til, om der kommer en hændelse fra `BiotopeViewer` (og implicit fra `InitializerDialog`). Den opretter altså biotopen med de valgte værdier fra `InitializerDialog` og sørger for at sætte `BiotopeViewer` og `BiotopeLogger` som observatører på biotopen. For at kunne gøre det skal den kende dem, og den har derfor en metode, `putViews`, som bliver kaldt i `Galapagos`, som kun har det formål at overføre observatorerne til `Controller`. Udover dette skal den styre slagets gang, dvs. den lytter til `BiotopeViewer` og sørger for, at `Biotope` kører det antal runder, der er valgt. Den bruger også en timer, så hver runde bliver opdateret, før biotopen kører næste runde.

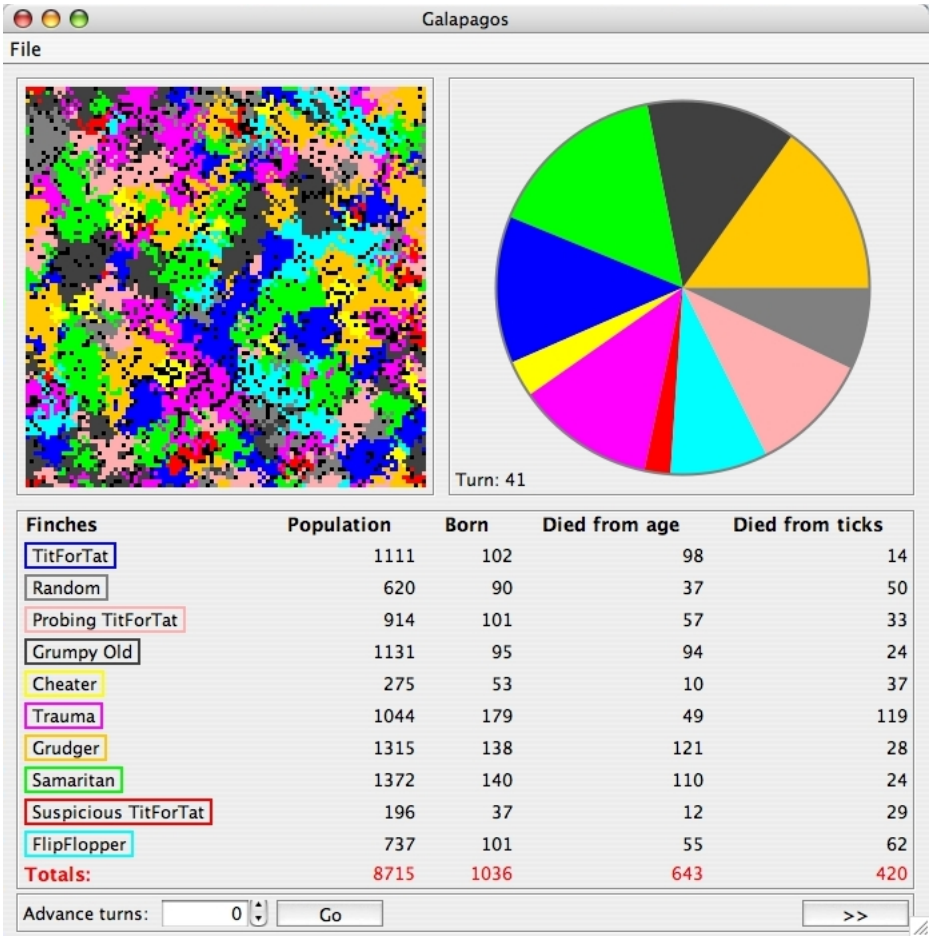
1.2.4 Klassen BiotopeViewer

`BiotopeViewer` står overordnet, for den grafiske brugerflade i vores program. GUI'en er struktureret vha. af et `GridBagLayout`, hvor vi har valgt at placere den grafiske datarepræsentation øverst, tekstrepræsentationen i midten og navigation i bunden. Valget af `GridBagLayout` som layout manager skyldes den øgede mulighed for kontrol med de indlejrede komponenters placering og justering ifht. hinanden.

Som udgangspunkt er det forsøgt at følge opgavens eksempel til design af brugerfladen. Dog har vi fundet det praktisk at tilføje nogle flere komponenter til både input- og outputdelen. Et kagediagram bliver tilføjet ved siden af vores `AreaPanel`, så man på et givent tidspunkt vil kunne få et hurtigt overblik over de individuelle finkers populationsandel i biotopen. Derudover har vi i stedet for de tre knapper til at køre trin valgt, at man manuelt kan indtaste x trin og derefter køre dem ved tryk på en `Go` knap. Vi ønsker også at kunne stoppe/starte vores model under kørslen, så en knap til dette bliver også indlagt i designet.
Eneste bug i GUI'en er, at toggle-knappen ikke automatisk deaktiveres, efter man har kørt et antal runder med Go, før man kan starte en automatisk kontinuert kørsel. Se screenshot af `BiotopeViewer` i Figur 2 på side 7.

1.2.5 Klassen PieChart

Denne klasse genererer vores kagediagram ved hjælp af tegnefunktioner fra `Java2D`-API'et. Praktisk bliver hver finkes populationsstørrelse repræsenteret ved et cirkeludsnit,



Figur 2: Screenshot af BiotopeViewer (vores brugerflade)

der hver især bliver tegnet som individuelle `Elipse2D.Double`-rektangler. Udsnittene opdateres efter hver tur, så de stemmer overens med den nuværende bestand af finker.

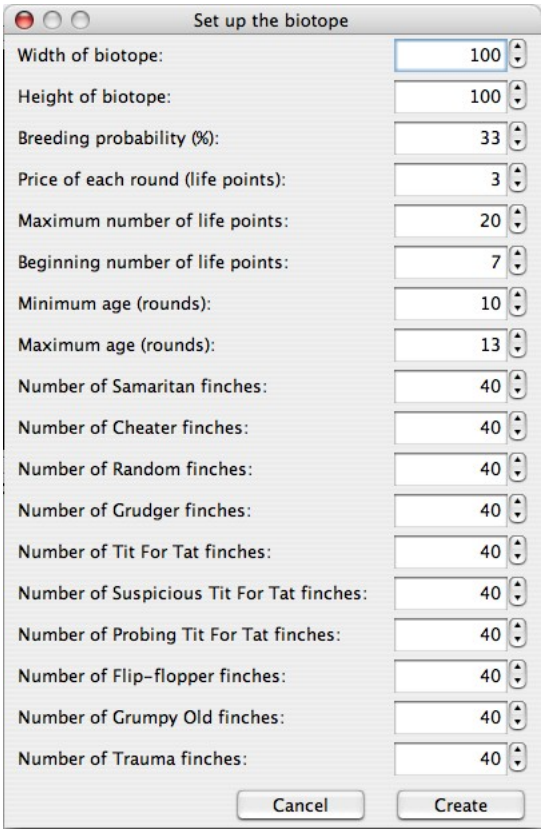
1.2.6 Klassen BiotopeLogger

Denne klasse implementerer brugerfladen `Observer` og er observatør på biotopen. Den har kun metoden `update`, hvor den henter information fra biotopen og laver statistikken.

Statistikken er ikke opsat, så den er nem at læse (for mennesker). Dette gør `BitopeViewer` i den grafiske brugerflade, og dette er derfor ikke nødvendigt i loggen. I stedet er loggen lavet, så det er meget nemt at få data ud af den. Dvs. man ud fra startoplysningerne om biotopen præcis kan regne ud, hvor de forskellige tal står. Dermed kan man, hvis man vil bruge loggen til at lave mere statistik over en kørsel, nemt parse informationerne.

1.2.7 Klassen InitializerDialog

For at få input til vores biotop har vi valgt at implementere en `JDialog`, hvor man ved hjælp af `JSpinners` kan indtaste data i fastsatte intervaller (vi har f.eks. valgt at begrænse biotopens størrelse til 1000x1000 af hukommelseshensyn). Ved et klik på Create-knappen valideres input vha. følgende betingelser: antallet af finker må ikke overstige biotopens størrelse, minimum age må ikke overstige maximum age, og start points må ikke overstige max life points. Er disse betingelser opfyldt, lukkes dialogen, og `Controlleren` overtager. Se screenshot af `InitializerDialog` i Figur 3 på side 8.



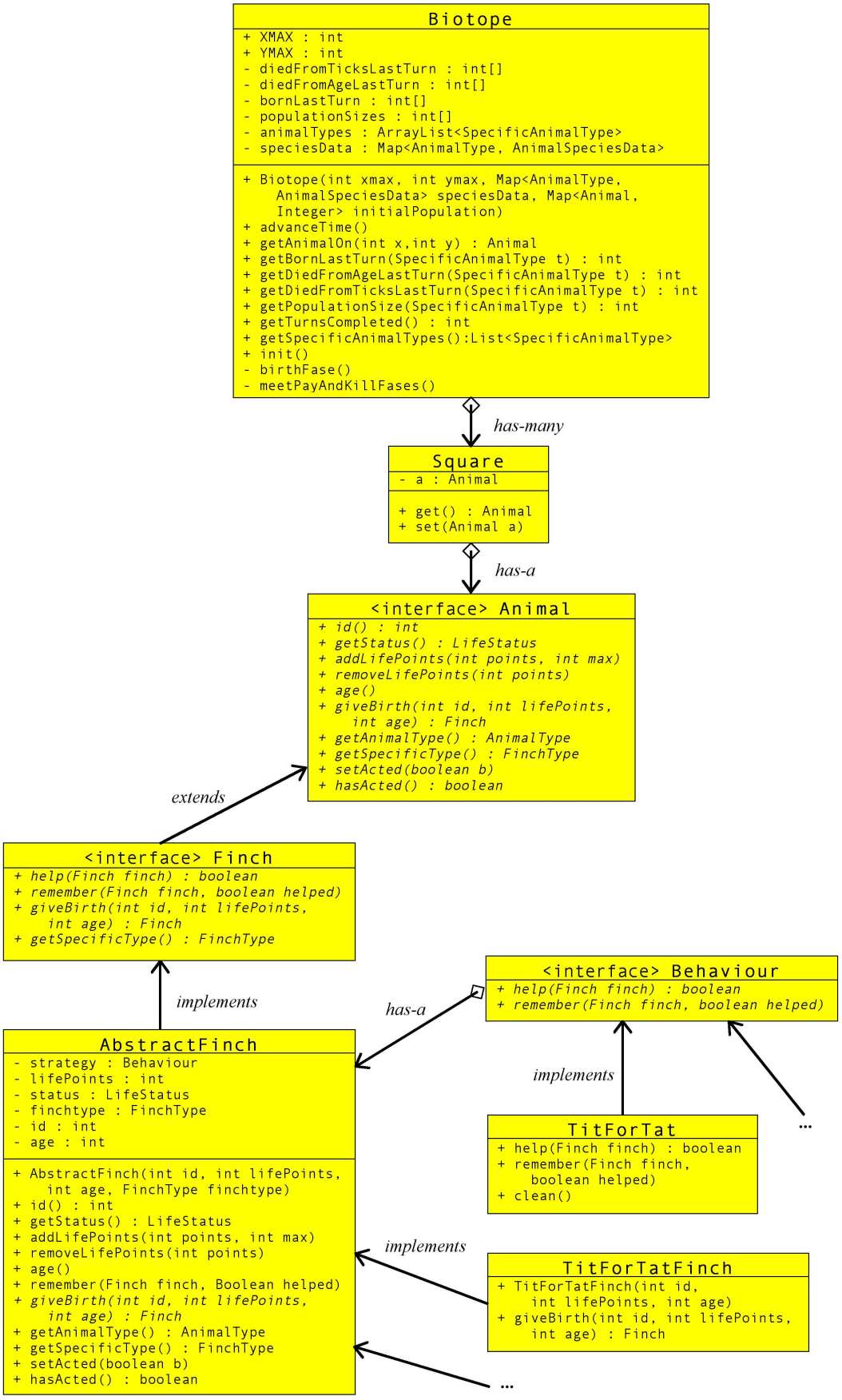
Figur 3: Screenshot af InitializerDialog

1.3 JUnit-tests

Vi har testet de forskellige finkers strategier, `AbstractFinch` generelt og en generel finketype (`SamaritanFinch`), `Biotope` og `BiotopeLogger`. Vi har vurderet, at det ville være formålsløst at teste dele af brugerfladen.
Alle tests går godt i Eclipse, som vi har brugt til design og udvikling. I DrJava kompilerer alt fint, men ingen tests vil køre af uvisse årsager. Da der ikke står specifikt i opgaveoplægget, at dette skal kunne lade sig gøre, har vi ikke brugt videre kræfter på dette.

2 Konklusion

Vi er endt op med et kørende program, der opfylder opgaveformulerinens krav. Vi har ud over dette desuden lavet egne finketyper og gjort det nemt senere hen at udvide modellen. Vores program kompilerer uden fejl, og alle tests validerer. Der er undervejs ikke umiddelbart fundet væsentlige fejl og mangler ud over de i rapporten nævnte småting, men dette ulukker selvfølgelig ikke eventuelle ikke-fundne fejl og bugs.



Figur 4: Oversigt over klasserne i modellen