Classes och Interfaces, Objects och References

Objekt-orienterad programmering och design

Alex Gerdes, 2018

Abstract class

- En abstract class är en class som inte kan skapa några objekt.
  - Syfte: Att samla gemensam kod för subclasses (undvika kod-duplicering).
  - Kan (men måste inte) ha metoder märkta `abstract` – dessa har ingen body, och subclasses måste göra *override* (om de inte själva är abstract).
  - Kan ha konstruktorer; dessa kan enbart anropas via `this()` i den egna klassen eller `super()` i subclasses.

Live demo – DrawPolygons

- …

Interface

- Ett interface (”gränssnitt” – men undvik den svenska termen) är en specifikation av ett antal metod-signaturer som tillsammans bildar en typ.
  - Enbart signaturerna ges – alla metoder är helt abstrakta.

- Interfaces i Java får specificera ”konstanter” – attribut som implicit är `public static final`, och som initialiseras till ett konstant värde.
  - Obs: används sällan. Grundprincipen för ett interface är en samling metod-signaturer.

Quiz: Abstract class vs interface

Vad är skillnaden på en abstract class med enbart abstrakta metoder, och ett interface?

1. En abstract class kan definiera attribut. Ett interface kan enbart definiera konstanter (`static final`).
2. En subclass kan bara ärva (*inherit*) från en superklass (ingen ”multiple inheritance”), men implementera många interfaces. I det avseendet är ett interface mer flexibelt.

Abstract class vs interface

- Använd det som passar avsikten bäst. Rule of thumb:
  - En abstract class representerar en generalisering av flera olika subclasses.
    - Mammal generaliserar Cow, Sheep, Cat…
    - Shape generaliserar Rectangle, Triangle, Circle…
    - …
  - Ett interface representerar en egenskap som kan delas av många (helt) olika classes.
    - Clonable representerar alla objekt som kan skapas exakta kopior av (implementerar clone()).
    - Runnable representerar alla objekt som kan köras som en separat thread (implementerar run()).
    - …
- Mer om när vi bör använda vad, och hur, senare i kursen.

Quiz (hard): Multiple inheritance

Varför tillåter inte Java så kallad multiple inheritance (dvs att en class kan ärva från flera superclasses)?

- Tänk er att vi har två classes : Cowboy och Artist.
- Båda classes har en metod draw() (med olika beteende).
- Tänk er att vi gör en class CowboyArtist som ärver från både Cowboy och Artist, och skapar ett objekt cowboyArtist.
- Vad händer när vi anropar cowboyArtist.draw()? Duellerande, eller akvareller?

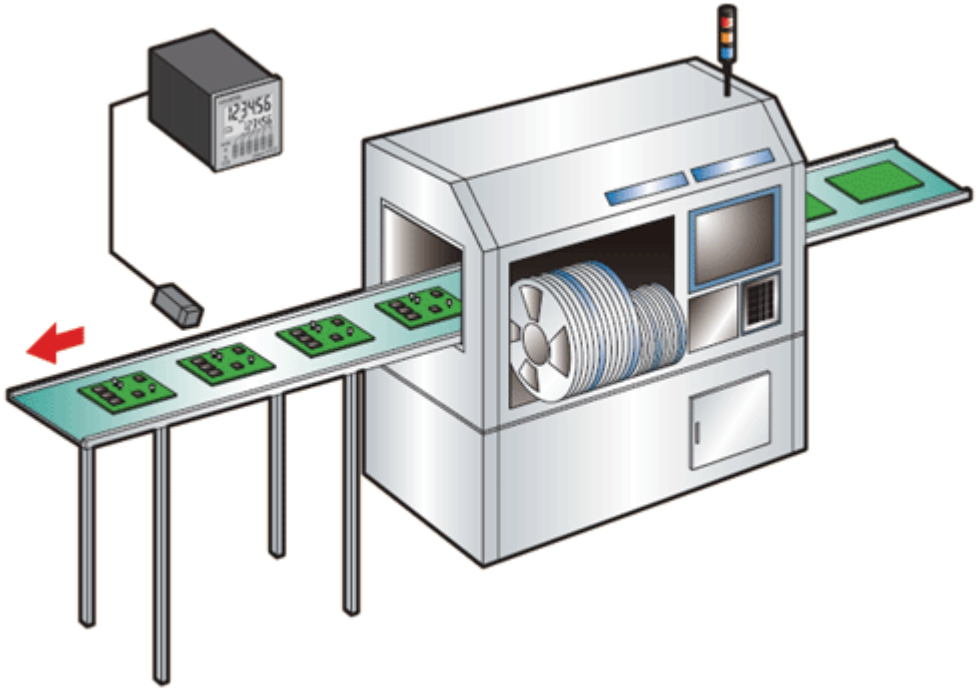
Class vs Object (static vs non-static)

Analogi: En Class är en produktionsmaskin. Maskinen är programmerad med ritningar för en sorts objekt, som den kan skapa efter önskemål (dvs anrop av konstruktor).

Maskinen kan ha ytterligare funktionalitet:

- Anrop av konstruktorer är meddelanden till maskinen att skapa nya objekt.
- Attribut märkta `static` tillhör maskinen, inte objekten den skapar.
- Anrop av metoder märkta `static` är meddelanden till maskinen, inte objekt.
- Allt som *inte* är märkt `static` (utom konstruktorer) är en del av ritningen för objekten som skapas.

Objekten vet alltid vilken maskin de kommer från (kan använda saker märkta `static`). Maskinen håller inte automatiskt reda på de objekt den skapat (kan inte använda saker som *inte* är märkta `static`).



Live demo

- …

Quiz

Hur kan vi både spara objektet i listan *och* returnera det? (Äta kakan och ha den kvar??) Har vi plötsligt två kopior?

- Objektet är vad det är. Det som sparas, och det som returneras, är en *referens* till objektet.

Referenser

- När vi tilldelar (assign) ett objekt till en variabel eller ett attribut så sparas en *referens* till objektet – inte objektet självt.
- Andra ord för referens är address eller pointer (pekare).

Värde (value) vs objekt

- Ett *värde* i Java har en primitiv typ.
  - E.g. `int` eller `float`

- En *literal* i Java är en syntaktisk representation av ett specifikt värde.
  - E.g. `42` eller `3.14159f`

- Ett *objekt* i Java är en komplicerad struktur med attribut (och metoder) inuti sig.

Quiz: primitive types?

Räkna upp de 8 primitiva typer som finns i Java.

- `byte` – 8-bit integer (-128/127)
- `short` – 16-bit integer (-32768/32767)
- `int` – 32-bit integer ($-2^{31}/2^{31}-1$)
- `long` – 64-bit integer ($-2^{63}/2^{63}-1$)

- `float` – 32-bit floating point
- `double` – 64-bit floating point

- `boolean` – `true` eller `false`, 32-bit
- `char` – 16-bit unicode character

Quiz: Den nionde primitiva typen

Det finns en ”hemlig” primitiv typ, vars värden du ofta arbetar med, men aldrig får se…

- Referenser!
  - Referenser (pekare till objekt) är små och enkla värden, adresser till en minnesarea där objekten bor.

Värden är små, objekt är stora

- Värden är små och enkla.
  - Inga värden överstiger 64 bitar (`long`, `double`).

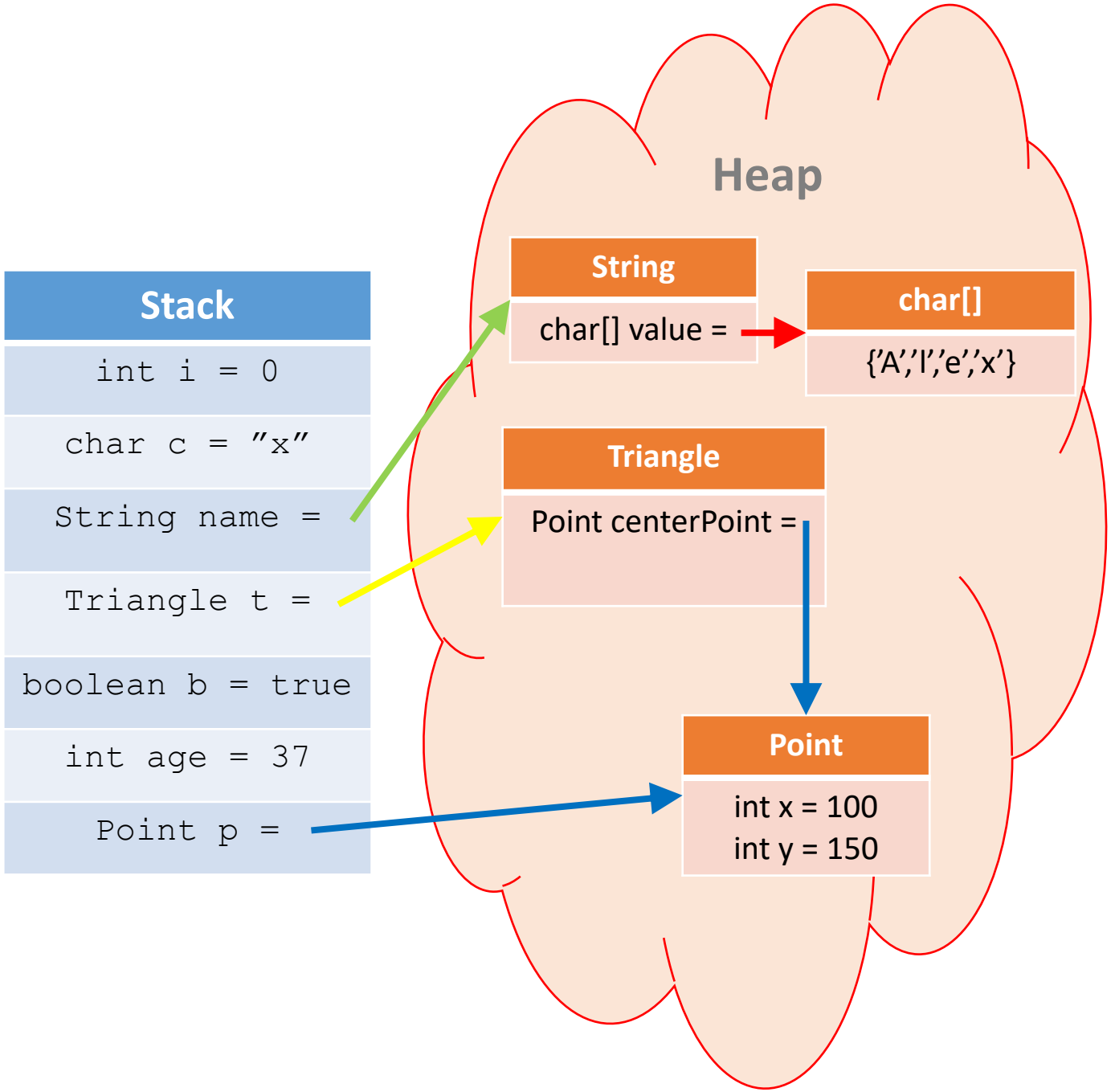
- Objekt är stora och kan vara hur komplexa som helst.
  - Innehåller attribut, som i sin tur håller värden.
  - Innehåller även en referens till ”ritningen”, där alla metoder är specificerade.

  - (”Ritningen”, dvs dess class, innehåller metoder, där varje *instruktion* (efter kompilering) är 1-2 bytes stor (Java byte code). Ett statement eller expression kompileras till en eller flera instruktioner. (Läs kursen ”Kompilatorkonstruktion” för mer info.))

Stack vs heap

- Värden för variabler sparas effektivt i en minnesarea kallad ”stack”.
- Objekt skapas och lever i en separat minnesarea kallad ”heap”.

- Namnen ”stack” och ”heap” kommer ursprungligen av de datastrukturer som används för att implementera dem (mer i kursen Datastrukturer).



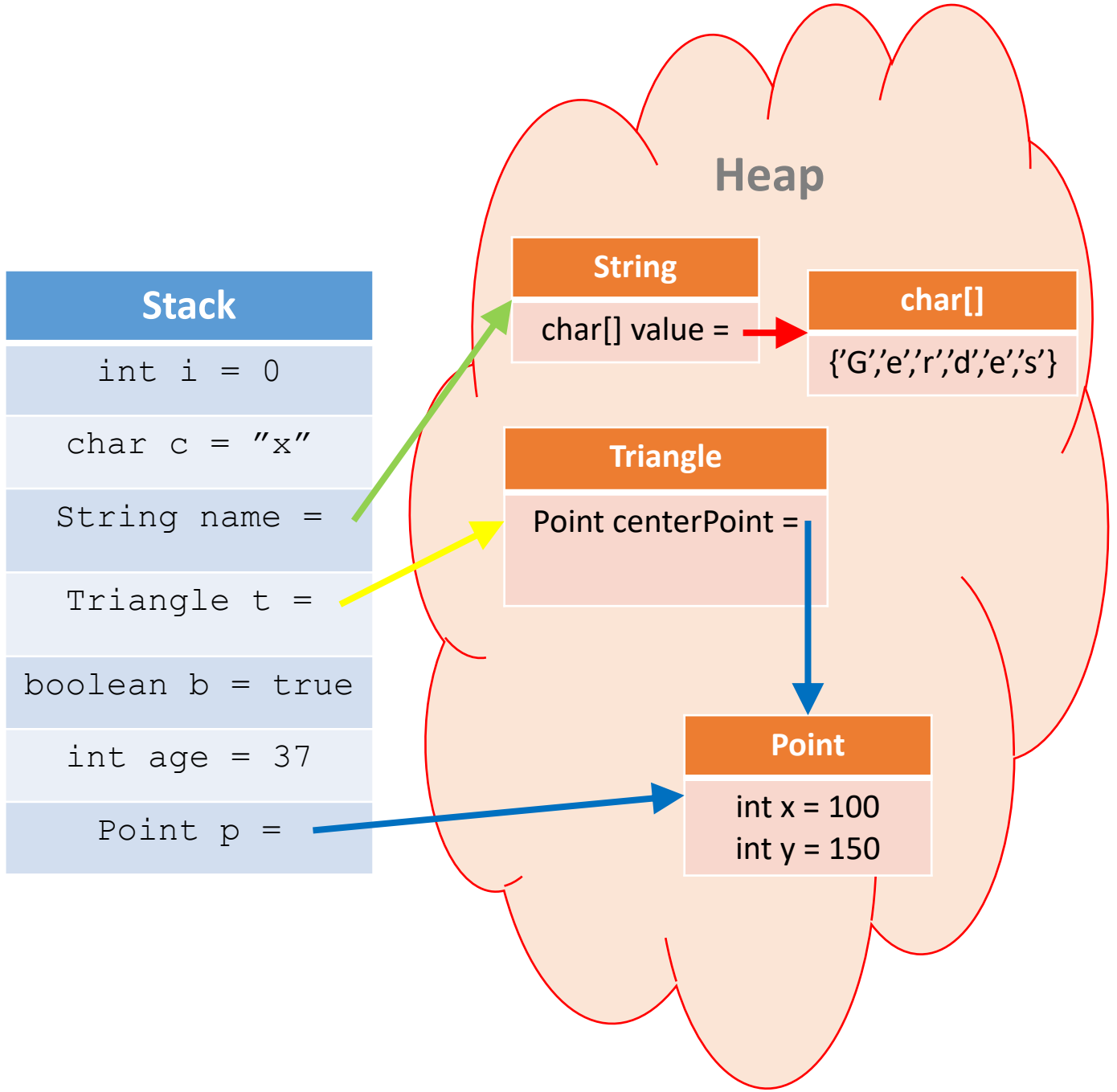
”Opaque pointers”

- Referenser (pointers) i Java är vad som kallas ”opaque” – dvs du som programmerare kan aldrig se deras värde, enbart arbeta med dem (e.g. anropa metoder via dem).

- Vissa andra språk – most famously C och C++ – har non-opaque pointers. Man kan då t ex addera en pointer med en int, och resultatet är en ny pointer som pekar någon annan stans i heapen.
  - Användbart för effektivitet vid explicit minnesanvändning – men totalt typ-osäkert!
  - Mer om detta i kursen ”Maskinorienterad programmering”.

Alias

- Två variabler eller attribut som håller *samma referens-värde* – dvs pekare till samma objekt i heapen – sägs vara *alias* för varandra.
  - Kan skapa förvirring ibland. Om vi t ex ändrar värdet av `p.x`, så kommer triangeln `t` att ha fått en uppdaterad mittpunkt, eftersom `t.centerPoint` och `p` är alias (referenser till samma Point-objekt).



Referenstyper

- Alla typer som inte är primitiva typer kallas *referenstyper* (reference types).
  - `String`, `Object`, `Runnable`, `char[]`, `ArrayList<Polygon>`, …
- Alla referenstyper har värden av samma sort: referenser.
- Typen för en referens används för att garantera att referensen i fråga pekar på ett objekt som kan uppföra sig som förväntat, i.e. har de metoder och attribut som förväntas för typen.
  - Mer om referenstyper, subtypning och polymorfism senare i kursen.

Summering

- Interface vs abstract class (vs non-abstract class)
- Static (hör till maskinen) vs non-static (hör till objekten den skapar)
- Värden (på stacken) vs objekt (på heapen)

- Djupare förståelse för referenser och hur de fungerar.

Studentrepresentanter

Reflektion

- Vad har jag lärt mig under den gångna veckan?

- Vad har varit mest förvirrande under den gångna veckan?