UML

Objekt-orienterad programmering och design

Alex Gerdes, HT-2018

UML

- Unified Modelling Language
  - Grafiskt modelleringsspråk för att beskriva olika aspekter av objekt-orienterade system.
  - Vi kommer att begränsa oss till UMLs klassdiagram – finns mycket mer.

Klassdiagram: Classes

- En class består av:
  - Dess namn
  - Dess variabler
  - Dess metoder (och konstruktorer)

+ = public
- = private
= protected
~ = none (i.e. package)

italics = abstract
underline = static
<<interface>>

Static? Abstract?

Customer
+ name : String # phoneNr : PhoneNumber
+ sendOrder() : void + receiveOrder() : void

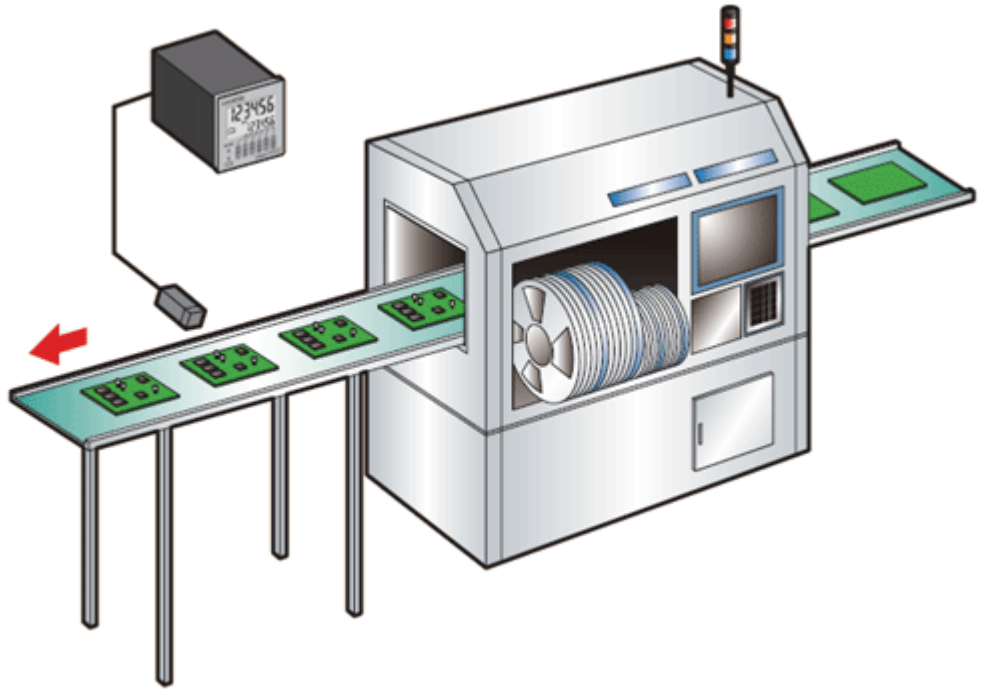
Class vs Object (static vs non-static)

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:

- 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` är en del av ritningen för objekten som skapas.

Objekten vet alltid vilken maskin de kommer från. Maskinen håller inte automatiskt reda på de objekt den skapat.



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.

- Representerar en abstrakt generalisering av flera olika subclasses:
  - Exempel: Mammal som abstract superclass till Cow, Sheep, Aardvark, …

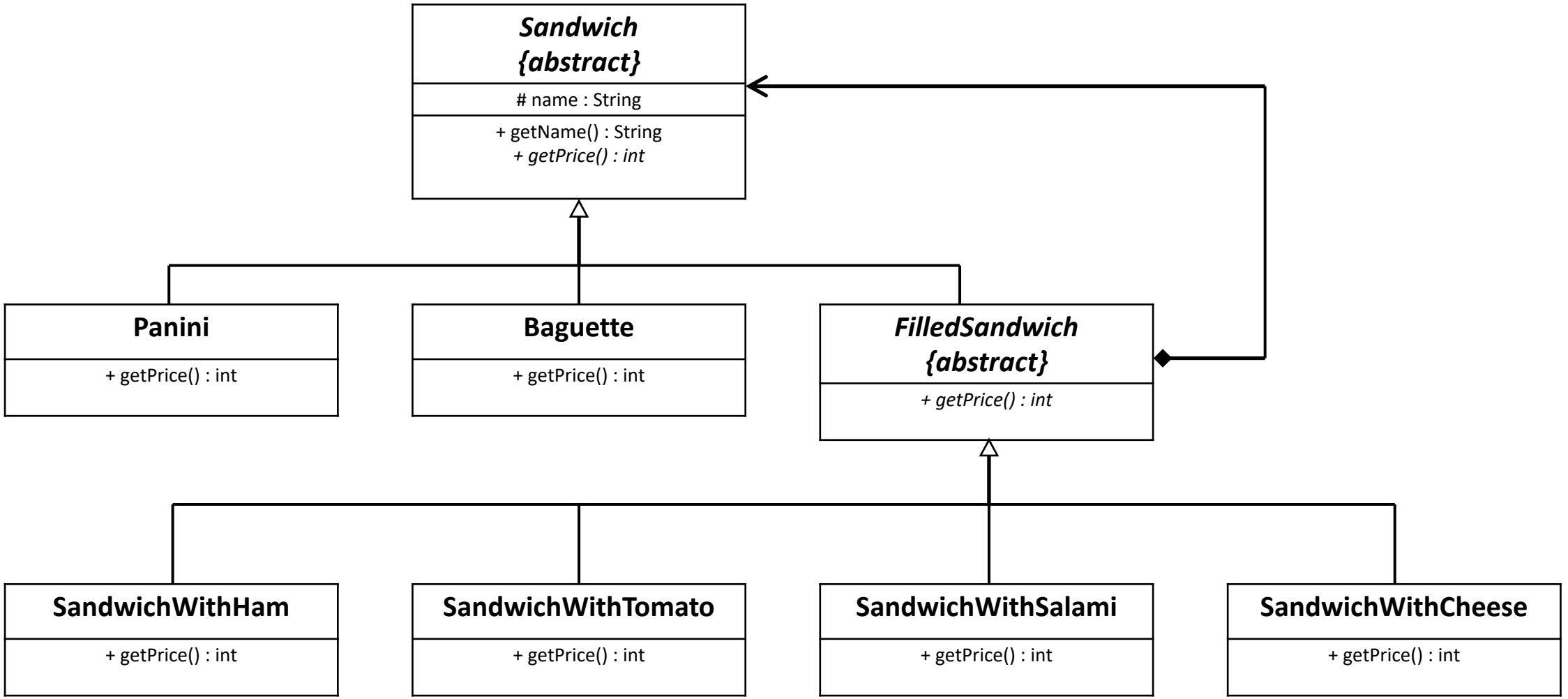
Klassdiagram: Relationer

- Fyra grundläggande typer av relationer mellan classes och interface:
  - Association (Has-A) 
    - En class har attribut (fields) som håller objekt av en annan class (interface).
  - Beroende (usage dependency) 
    - En class (interface) använder eller skapar objekt av en annan class (interface).
  - Generalisering (Is-A) 
    - En class (interface) är en direkt subclass (subinterface) till en annan class (interface).
  - Realisering (implements) 
    - En class implementerar ett interface.

Klassdiagram: Relationer specialfall

- Specialfall av association:
  - Aggregation (parts-of-a-whole) 
    - En class består av en eller flera delar, som representeras av en annan class.
  - Komposition (subpart-of) 
    - En starkare form av aggregation, där ”delen” bara får förekomma i en ”helhet”, och existerar endast så länge dess helhet existerar
  - Inkapsling (encapsulation) 
    - En class existerar inuti en annan class (vi kommer till inner classes senare).

Klassdiagram: Exempel



Övning

- Utgå från ”DrawPolygons.java” (från tidigare övning)
- Syfte: Refactor till en design som följer OCP. Öva UML.
- Utför följande refactoring steps:
    1. Skapa en ny class Polygon som håller ihop name och centerPoint. Skapa en constructor Polygon(String,Point). Använd i DrawPolygons.
    2. Ge Polygon en paint-metod. Flytta relevant funktionalitet från DrawPolygons.paint.
    3. Förenkla if-satsen i Polygon.paint genom att lyfta ut relevant funktionalitet i tre separata metoder: paintSquare, paintTriangle och paintRectangle.
    4. Skapa tre subklasser till Polygon: Square, Triangle och Rectangle. Ta bort alla aspekter av name. Ge varje subklass rätt paint-metod. Förenkla paint-metoden i DrawPolygons.
    5. Gör Polygon abstract. Gör constructor Polygon(Point) privat. Skapa en public constructor Polygon(int,int) som anropar Polygon(Point), och förenkla i DrawPolygons.
- För varje delsteg, rita upp det klassdiagram (UML) som beskriver systemet.
- Skriv JavaDoc för alla klasser och metoder
- För extra utmaning:
    - Fundera över andra sätt (än att lägga till fler sorters polygoner) som programmet kan komma att utökas eller förändras. Hur kan ni planera för dessa?