Subclassing vs Delegation

Objekt-orienterad programmering och design

Alex Gerdes / Joel Hultin, 2017

Implementation inheritance

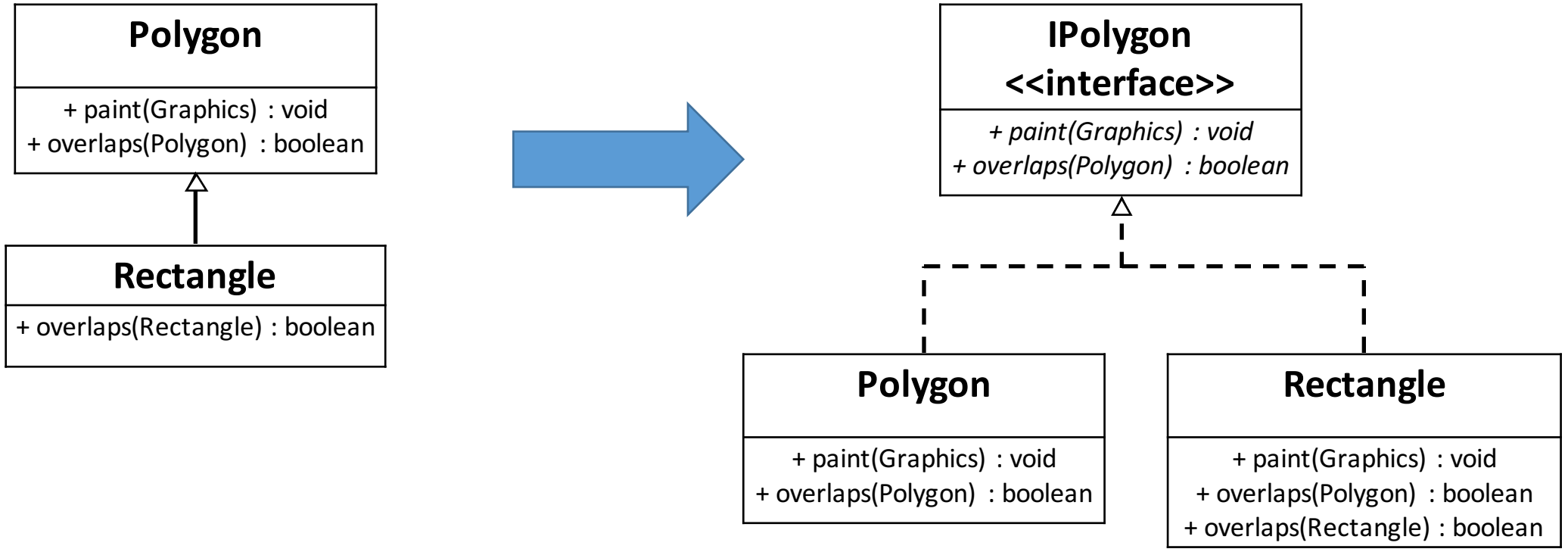
- Subclassing, eller *implementation inheritance* (implementationsarv), ger oss två fördelar:
  - Polymorfism
    - Ett objekt av en subclass kan användas som om det var ett objekt av sin superclass.
    - Ger flexibilitet, ”Plug-n-play”; kan e.g. arbeta med listor av objekt med superklassen som statisk typ, som vid run time kan vara av antingen superclass eller (någon) subclass.
  - Återanvändning av kod (code reuse)
    - Definiera gemensamma metoder i superklassen som kan ärvas av dess subclass(es).
    - Don’t repeat yourself, ingen cut-n-paste.
    - Ger ett enda ställe för förändring vid behov – maintainability.

Quiz: Polymorfism reclaimed

- Om vi inte hade implementation inheritance (dvs subclasses), hur kan vi ändå åtnjuta fördelarna av polymorfism?

- Svar: Med hjälp av interface inheritance.
  - Skapa ett interface som specificerar de relevanta metoderna.
  - Låt både ”superclass” och ”subclass” implementera detta interface.

Polymorfism reclaimed

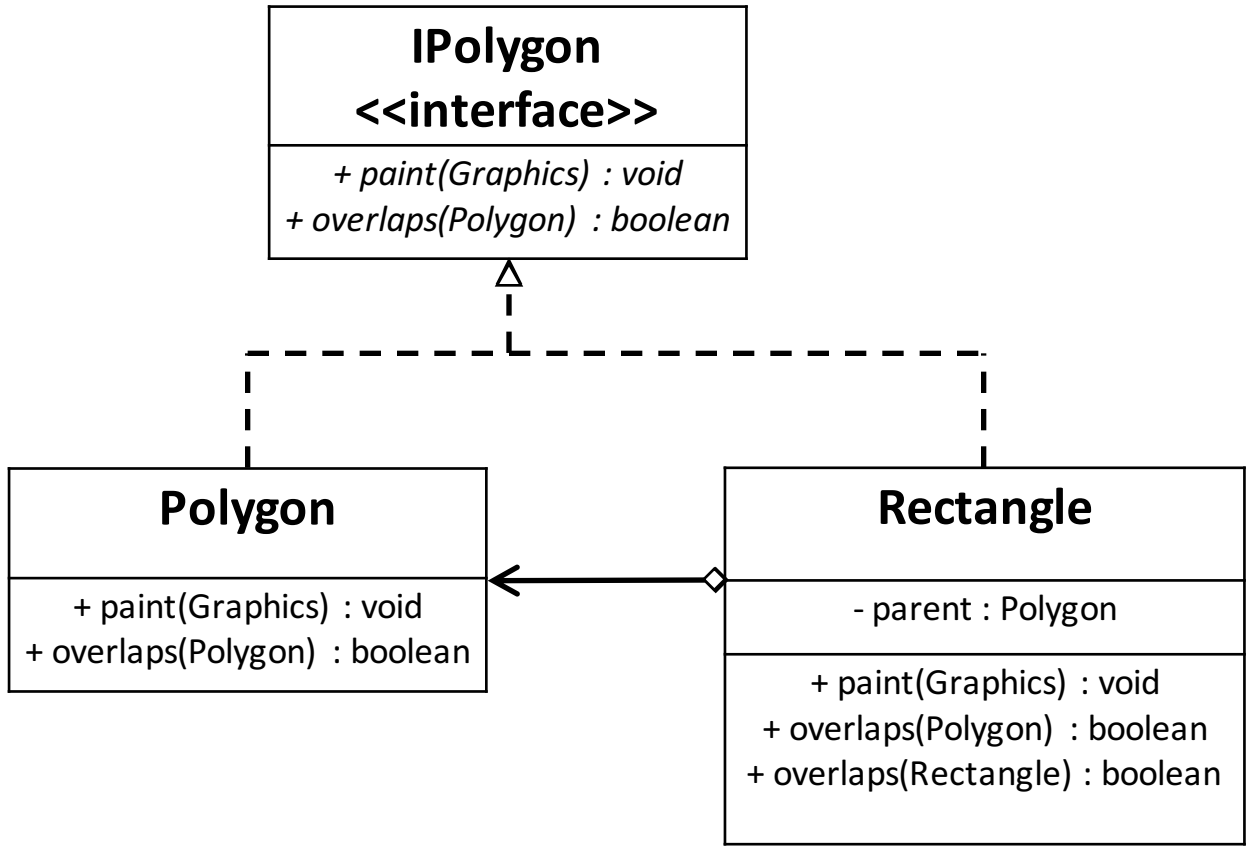


Quiz: Code reuse reclaimed

- Om vi inte hade implementation inheritance (dvs subclasses), hur kan vi ändå åtnjuta fördelarna av code reuse?

- Svar: Med hjälp av komposition/aggregation.
  - Låt ”subklassen” ha en referens till ett objekt av ”superklassen”. Återanvänd metoder hos ”superklassen” genom explicita vidareanrop – kallas *delegering*.

Code reuse reclaimed




```
class Rectangle {

  private Polygon parent = …

  public boolean
    overlaps(Polygon other) {
    parent.overlaps(other);
  }

}
```

”Ärvd” metod via delegering

Inheritance vs delegation

- Implementation inheritance existerar, så vi måste inte ”reclaima” fördelarna på annat sätt.

- Implementation inheritance är dock inte alltid rätt val. Delegering har många egna fördelar, och kan alltid väljas istället.
  - Fokus för dagens övning.

Övning

- Implementera följande tre scenarier, dels med subclassing, dels med delegering:
  - En plattform för att spela RPGs (role playing games) har ett koncept av användare (`User`). I olika spel kan en användare vara antingen spelare (`Player`) eller spelledare (`GameMaster`). Det finns gemensam kod (`userMethod()`), och kod som är specifik för de olika rollerna (`playerMethod()` och `gameMasterMethod()`).
  - En kvadrat är bara en speciell sorts rektangel. Implementera en `Rectangle` (oberoende av tidigare uppgifter) med höjd och bredd, med setter och getter methods för båda, och en kombinerad `setSize(int h, int w)`. Implementera en `Square` med den extra (overloaded) metoden `setSize(int w)`.
  - En skrivare kan använda antingen laser eller bläck. Implementera en `Printer` med metoden `print()`. Implementera en `LaserPrinter` med metod `changeToner()`, och en `InkPrinter` med metod `changeInk()`.

- Diskutera för varje scenario för- och nackdelar med subclassing vs delegering. Bör subclassing användas?
  - Rita upp UML-diagrammen för varje fall.
  - Försök hitta generella principer för när subclassing är lämpligt.