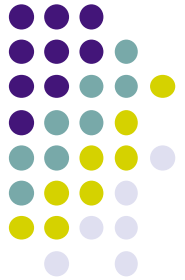
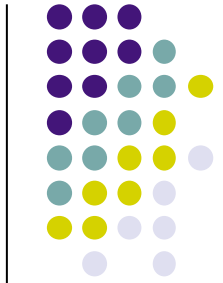
Final remarks



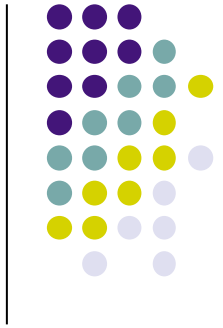
- This completes the part of the course related to data abstraction
  - Explicit state and object-oriented programming
  - Java, multiple inheritance, and exceptions
- We have covered three of the four themes
  - Functional programming (including recursion, invariant programming, and higher-order programming)
  - Language semantics (a complete operational semantics)
  - Data abstraction (including explicit state and object-oriented programming)
- We end this theme with a reflection on language design and an introduction to concurrent programming

Java, Scala, and language design



- We have discussed some of the principles that were used to design Java (1990s)
  - True data abstraction (encapsulation, GC)
  - Almost all entities are objects
  - Support for object-oriented design
- Scala has added two principles to this (2000s)
  - Strict separation between mutable/immutable
  - Everything is an object (including functions)
- These principles considerably increase Scala's expressive power compared to Java
  - We consider that Scala is a worthy successor to Java

Final theme: concurrency



- The final theme of the course will be concurrency
  - Multiple activities that evolve independently and collaborate
  - There are three fundamental forms of concurrent programming: deterministic dataflow, message passing, and shared state
  - All three were invented (or discovered?) in the early 1970s!
- We will present deterministic dataflow in depth
  - It is an extremely powerful yet easy to use model that deserves to be more widely known
  - All the techniques of functional programming generalize for deterministic dataflow