Design for Reuse

Delegation:

- Delegation is simply when one object relies on another object for some subset of its functonality
  - Sorter is delegating functonality to some Comparator implementation
- Judicious delegation enables code reuse
  - Sorter can be reused with arbitrary sort orders
  - Comparators can be reused with arbitrary client code that needs to compare integers

Delegation and design:

- Small interfaces
- Classes to encapsulate algorithms
  - ex: the Comparator, the Strategy pattern

Inheritance:

- Typical roles:
  - An interface defines expectations/commitment for clients
  - An abstract class is a convenient hybrid between an interface and a full implementation
  - A subclass overrides a method definition to specialize its implementation

Benefits of Inheritance:

- Reuse of code
- Modeling flexibility
- A Java aside:
  - Each class can directly extend only one parent class
  - A class can implement multiple interfaces

Power of object oriented interfaces:

- Subtype polymorphism
  - Different kinds of objects can be treated uniformly by client code
- e.g., a list of all accounts
  - Each object behaves according to its type
- If you add new kind of account, client code does not change

Inheritance and subtyping:

- Inheritance is for code reuse
  - Write code once and only once
  - Superclass features implicitly available in subclass
- Subtyping is for polymorphism
  - Accessing objects the same way, but getting different behavior
  - Subtype is substitutable for supertype

Java details: final:

- A final field: prevents reassignment to the field after initialization
- A final method: prevents overriding the method
- A final class: prevents extending the class
  - e.g., public final class CheckingAccountImpl

Type Casting:

- Sometimes you wan a different type than what you have
  - ex: float pi=3.14; int indianapi=(int) pi;
- Useful if you have more specific subtype
- Advice: avoid downcasting types

  - Never downcast within superclass to a subclass

Behavioral Subtyping:

- Compiler enforced rules in Java:
  - Subtype (subclass, subinterface, object implementing interface) can add but not remove methods
  - Overriding method must return same type or subtype
  - Overriding method must accept same parameter types
  - Overriding method may not throw additional exceptions
  - Concrete class must implement all undefined interface methods and abstract methods
- A subclass must fulfill all contracts its superclass does:
  - same or strong invariants
  - same or stronger post
  - conds for all methods
  - same or weaker pre
  - conds for all methods

Parametric polymorphism via java generics:

- Parametric polumorphism is the ability to define a type generically to allow static type
- checking with out fully specifying types
- The java.util.Stack instead
  - A stack of some type T:

```
public class Stack<T> {
   public void push(T obj){...}
   public T pop() { ...}
}
```

- Improves typechecking, simplifies client code

Notes:

- Template method design pattern
- Decorator design pattern