

Design with Inheritance

Designing with Inheritance

- Initial design of an object oriented system should use inheritance where appropriate.
 - ... and only where appropriate.
- First requirement: an “Is-A” relationship

```
class X extends Y {...}
```

 - should be able to say “Every X is a Y.”
 - or “X is a more specific type of Y.”

Reuse

- Inheritance allows easy reuse of code.
 - can reuse code from the parent class
 - ... or several related classes can inherit a common parent
- Look for classes that could be reused in future work or when expanding the code.
- Move code up in the class hierarchy.
 - ... so it can be reused in related classes.

Abstract Classes

- There may be classes in your class hierarchy that should never be instantiated directly.
 - ... but should be subclassed and the children instantiated.
 - eg. `Person` in the blackjack design, `Shape` in the drawing program design.
- These are “abstract classes”.
 - classes that are only there to be inherited

Abstract Classes

- Classes can be marked as abstract in Java:

```
abstract class Shape { ... }
```

 - The `Shape` class can not be instantiated, but could be inherited.
- Abstract classes can be specified to explicitly mark that they are only for code reuse.

Overriding Variables

- It's possible to override variable definitions in a subclass. (“shadow” variables)
 - Don't, unless you **really** know what you're doing.
- New definitions (with different type) will also be used by non-overridden methods.
 - Results could be very different than what you expect, depending on implementation details.
- Defining new variables is okay.
 - ... and often necessary to capture new behaviour.