# Inheritance and Polymorphism

Saturday, 20 January 2018 12:23 PM

## Interfaces

Contract between two sides of the abstraction barrier

- Syntactic contract, not semantic contract
    - Method exists, but naming may not be indicative
- Methods are automatically "public"

`@Override`

- Annotation to indicate implementation of interface methods

`implements`

- Class can implement more than one interface
- Class can implement more methods than in interface
- Must have same method signature and return type

### Interfaces as types

- Reference type
- Cannot be instantiated
- Assigning variables to objects: only if object is instance of variable type
    - E.g. Circle is instance of Circle, GeometricShape and Printable
- Shape s = Circle c; s does not have moveTo() method at compile time

`java.awt.color`

## Polymorphism

- C = static/ early binding: determined at compilation time
- Java = methods have dynamic/ late binding, dynamic dispatch (polymorphism)
    - Instructions determined at run time depending on instantiation
- DRY: Don't Repeat Yourself

## Inheritance

- Common fields in a parent class?

`extends`

- Subclasses have no access to private fields in parent class
- Need to call constructor of parent
- super(...);

`instanceof`

`protected`

- Subclasses can access protected fields and methods
- Standard access modifier for constructor

### Overloading

- Looks at method signature (name + type/order of arguments)
    - Return type not in signature
- Can have two methods with same signature, in super and subclass (overrides superclass)

## Object class

`equals(Object obj)`

- Checks for same reference, not semantic equality (have to override)
- `hashCode()`
    - Overrided with equals()

`toString`

- Automatically called during string concatenation

## Exercises

1. Line 2
    - `incompatible types: Shape cannot be converted to Printable`
    - `Printable c2 = (Printable) c1;`
        - Explicit casting needed so that code can compile