O04 Inheritance 1

Inheritance
Hiding and overriding
Polymorphism
The super keyword

Inheritance

A class that inherits is known as a *subclass*, *derived class*, or *child class*. Its parent is known as a *superclass*, *base class*, or *parent class*.

- Subclasses inherit via the `extends` keyword
- All classes implicitly inherit from `java.lang.Object`

Overriding and Hiding Methods

- Instance methods
  - If method has same signature as one in its superclass, it is said to **override**. Mark with `@Override` annotation.
  - Same name, number and type of parameters, and return type as overridden parent method.
  - The type of the instance (not the variable referencing it) determines the method.
- Class methods
  - If it has same signature, it **hides** the superclass method.
  - The class with respect to which the call is made determines the method.

Polymorphism: “Many-forms”

A reference variable may refer to an instance that has a more specific type than the variable.

The method that is called depends on the type of the instance, not the type of the reference variable.

This overriding of methods is a form of **runtime polymorphism** (actual underlying type will dynamically determine the behaviour). Interfaces also provide a form of runtime polymorphism.

Method overloading (same name, different type signatures) and operator overloading (e.g., +) are a form of **compile-time polymorphism**.

Hiding Fields

When a subclass uses a field name that is already used by a field in the superclass, the superclass' field is **hidden** from the subclass.

Hiding fields is a bad idea, but you can do it.

The `super` keyword

You can access overridden (or hidden) **members** of a superclass by using the `super` keyword to explicitly refer to the superclass.

You can call superclass constructors by using `super`() passing arguments as necessary.