O04 Inheritance

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 modifiers, return type, name, and sequence of parameter types as the overridden parent method.
  - **Dynamic dispatch**: The type of the object (not the variable referring to it) determines which method is called.
- 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**.

The Object superclass

All Java classes ultimately inherit from **one** root class: `java.lang.Object`.
Some of its methods are:

- `clone()` returns (shallow) copy of object
  - Note: cloning is not automatically supported by all classes.
- `equals(Object other)` establishes semantic equivalence
- `finalize()` called by GC before reclaiming
- `getClass()` returns runtime class of the object
- `hashCode()` returns a hash code for the object
- `toString()` returns string representation of object

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.

Type Casting

A reference to an object of a given class can be explicitly converted to a reference to a subclass: this is called (dynamically) “type casting”.

Because it is not guaranteed that the object is of the subclass, explicit casting can always result in a `ClassCastException`, which must be caught.

```
Try {
  SubClass y = (SubClass)x;
catch (ClassCastException e) {
  // statements to execute if x is not of class SubClass
}
```