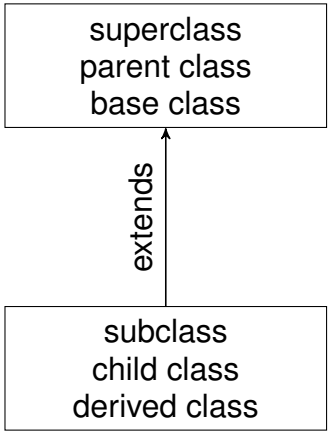
Inheritance

04

- Inheritance
- Polymorphism
- Class fields

Inheritance

- Java classes form a hierarchy:
  - subclasses extend their superclass.
- A variable (expression) of a class type can hold (evaluate to) a reference to an object of any subclass (or subsub-, or sub...subclass)
- `java.lang.Object` is the root of the class hierarchy (ultimate ancestor of every class).



Defining a subclass

```
public class Row extends Region {
  // ...
}
```

- Keyword `extends` followed by superclass name.
- In Java, every class (exept `Object`) has *exactly one* immediate superclass:
  - At most one `extends` declaration in a class definition.
  - If not specified, implicitly `extends Object`.

- The subclass inherits all non-private fields and methods of the superclass.
  - That is, these exist in the subclass even if not declared.
- The subclass can override a method from the superclass, by declaring a method with the same signature and an alternate implementation.
  - The default `toString` and `equals` methods are inherited from `Object`.

Dynamic dispatch

- Reminder: A variable (expression) of a class type can hold (evaluate to) a reference to an object of any descendant class.
- When an instance method is called on an object, the method that is called is *determined by the object's class*, not the type of the expression (variable) that yielded the object.
- This gives rise to run-time polymorphism.
- A class can declare several methods with the same name, but different parameter types: which is called is determined by the argument's types (compile-time polymorphism).

Keyword `super`

- Within a subclass, prefixing a (method or field) name when used with `super.` makes it refer to the superclass' method, even if overridden by the subclass.
- Within a subclass constructor, `super` can be used to call a superclass constructor:

```
public Row(int i, int n) {
  super(i, i, 0, n);
}
```

- If a subclass constructor does not explicitly call a superclass constructor, it implicitly calls a superclass constructor with no parameters.

Type casting

- A reference to an object of a class can be explicitly converted to reference an object of a subclass.
- This can *fail* (raise an exception) if the object is not of the subclass.

```
try {
  Subclass y = (Subclass)x;
  // executes if x is of type Subclass
}
catch (ClassCastException e) {
  // executes if x is not of type Subclass
}
```

Design considerations

- Composition vs. inheritance (*has-part* vs. *is-a*).
  - A class can have fields that are objects.
  - A composite class can provide functionality of several component classes by delegating to them.
- Different super-/subclass relationships:
  - subclass extends superclass (adds fields/functionality);
  - subclass restricts superclass (simplifies interface);
  - subclasses provide different (exclusive) implementations of a common interface: the superclass may be declared `abstract`, and have abstract methods (without body).

Class fields and methods

- A field declared `static` belongs to the *class*, not object.
  - Class fields are shared (accessible) by all instances of the class.
  - Class fields are accessible by static methods.
- A subclass can override superclass static fields or methods; this *shadows* the superclass field/method within the subclass.
- Static dispatch: when a static method is called, the method that is called is determined by the type of the referring expression.