LUND UNIVERSITY

Topic 4: Inheritance

Extending, Overriding and Final

Jesper Pedersen Notander

Department of Computer Science
Lund University Faculty of Engineering

Learning Scala
Seminar 2

Contents

- Abstract and concrete classes
- Simple inheritance
- Overriding members
- Polymorphism and dynamic binding
- Keywords: `abstract`, `extends`, `override` and `final`

Abstract and Concrete Classes

- *Concrete classes*, *"normal classes"*
  - *Defines* members, e.g. `val x: Int = 0;`
  - Can be instantiated
  - Must define inherited *abstract* members.
- *Abstract classes*
  - Defines members
  - Cannot be instantiated
  - *Declares* abstract members, e.g. `val x: Int;`
- Abstract classes are implemented by adding the keyword `abstract` to the class declaration, e.g.

```
abstract class MyAbstractClass
```

Abstract Members and Declarations

```
abstract class C(val p1: Int, private var p2: Int) {
    protected var f1: Int

    private def m1(a: Int): Int
    def m2: Int
}
```

- Fields and methods are declared in the body or the parameter list of the class.
- Methods can be declared without brackets if they have no arguments. Thus, `def m2: Int` <=> `val m2: Int`, when accessing point of view.
- The access modifier defines the visibility of the declaration:

public	Visible where the class is accessible, default (no modifier)
protected	Visible in the class and its subclasses
private	Visible in the class

Extending a Class

- To inherit from a class place the keyword `extends` after the class name in the declaration of a class, followed by the name of the class to inherit from, e.g.

  ```
  class B(x: Int)extends A(x)...
  ```

  Terminology: class B *inherits* class A, type B is a *subtype* of type A, class A is a *superclass* of class B and class B is a *subclass* of class A
- All members of the superclass become members of the subclass except:
  - Private members
  - Members with the same name as a member in the subclass.

Overriding Members

- When a subclass defines a member declared in a superclass the subclass is said to *override* the member.
- Methods and fields share the same namespace and it is allowed to override a field with a method and vice versa.
- The modifier `override` is required when overriding a concrete member, e.g.

```
override def myfun(x: Int)= 0
```

Example Overriding

```
abstract class A {
  def attr: Int
  def fun: Int = 1
}

class B extends A {
  def attr = 0          // concrete definition of A.attr
  override def fun = 2 // override of abstract A.fun
}

// override of abstract method A.attr with the field attr
class C( val attr: Int) extends A
```

```
scala> (new B).attr
res13: Int = 0
scala> (new B).fun
res14: Int = 2
scala> (new C(10)).fun
res16: Int = 1
scala> (new C(10)).attr
res17: Int = 10
```

Preventing Overrides

- Members can be prevented from being overridden by subclasses by adding the `final` modifier.

```
class A { // subclasses of A cannot override x
  final def x = 1
  def y = 2
}
```

- A class can be prevented from being subclassed by placing `final` in its declaration.

```
final class B { //class B cannot be subclassed
  def x = 1
  def y = 2
}
```

Subtyping Polymorphism

```
class A {
  def f = 1
}

class B extends A {
  pverride def f = 2
}

val a: A = new A()
val b: A = new B() // polymorphism
```

- Polymorphism in this presentation refers to *subtyping polymorphism*, which is the ability of a language to define subtypes and refer to values of them using a reference with a supertype-

Dynamic Binding

- The other part of polymorphism is *dynamic binding*, which means that the method that is actually invoked when access is determined by the runtime object of the object, e.g. `a.f =>1` whereas `b.f =>2` although both a and b are of type A.

Summary

- Abstract classes are declared with the keyword `abstract` and can contain abstract members.
- Abstract members are fields or methods that are declared but not defined in the class, e.g. `def fun: Int;`
- Inheritance is done by placing the keyword `extends` after the name in a class declaration, followed by the name of the super class, e.g. `class B(args...)extends A(args...)`
- A member overriding an inherited member must be declared with the `override` modifier, except if the inherited member is abstract.
- To prevent a method or a class to be overridden or subclassed the modifier `final` can be placed in the declaration, e.g. `final class C` or `final def fun = 0`
- Subtyping polymorphism is the ability to create subtypes and refer to them with their super type. Method calls are bound dynamically to the method in the runtime type of an object.