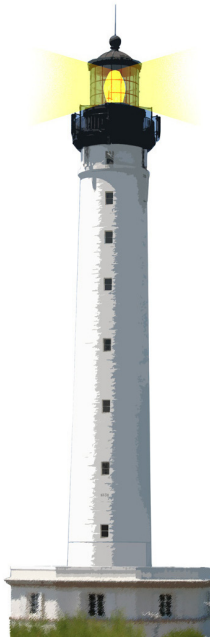
Learning Object-Oriented Programming and Design with TDD

Type aspects of Java

S. Ducasse

http://stephane.ducasse.free.fr



Pharo

http://www.pharo.org

Objectives

- Understanding dynamic and static
- Casts

Static vs. Dynamic Types

```
A a = new B();
```

- The static type of variable a is A i.e., the statically declared class to which it belongs.
  - The static type never changes.
- The dynamic type of a is B i.e., the class of the object currently bound to a.
  - The dynamic type may change throughout the program.

```
a = new A();
```

Now the dynamic type is also A!

Static vs. Dynamic Types

This works too with method signature

```
public class A { }
public class B extends A { }
```

```
public class Main {
  public static void main(String[] args) {
    dynclassOutput (new B());
    dynclassOutput (new A());
  }
  public static void dynclassOutput (A a) {
    System.out.println(a.getClass().getName());
  }
}
```

What is the static / dynamic type of a there?

Overloading is a bad idea

You can have multiple methods with the same name and types argument

```
visit (ProgramNode n) {}

visit (Assignment n) {}

visit (SequenceNode n) {}
```

- Avoid it as much as possible... it makes code less extensible
- Overloading makes your code difficult to understand in presence of subtyping
- Programming in OOP is using subtyping
- Check other lectures

About Casts

- Tell the compiler that the type of the object is another one...
- you cannot downcast to a type lower than the runtime type of the instance.

```
B b = new B();
A upcasted = (A) b;  <=== this is the "upcast"
B downcasted = (B) upcasted; <=== this is the downcast
```

About Casts: the runtime errors

```
A upcasted = new A();
B downcasted = (B) upcasted; <=== this is an invalid downcast that compiles but does
      not run
```

- but it does not run
- because at runtime it checks "is the object a B?" NO => boom
- a runtime check is necessary
- You cannot downcast to a lower type than the object dynamic type

The case of equals

The only place where you should use a downcast is equals(Object o)

```
public class Person
{
   String firstName;
   String lastName;
   public String getFirstName() { return firstName; }
   public String getLastName() { return lastName; }

   @Override
   public boolean equals(Object o) {
      if(o instanceof Person)
         return isSamePerson((Person)o);
      return false;
   }
  public  boolean isSamePerson(Person o) {
      return firstName.equals(o.firstName) && lastName.equals(o.lastName);
   }
}
```

The case of equals

In the case of equals, it's to compare this with the parameter. The signature says the parameter that represents the other object that you will compare yourself against is an instance of Object, then in your equals, if you need to compare specific result of methods (or fields), you need to downcast

```
Class B {
  public boolean equals(Object o) {
    if (this == o) return true;
    if (! (o instanceof B)) return false;
    B other = (B) o;
    // Do stuffs with other and this to compare getter/fields...etc
}
```

Kind of Summary

- Static types are known by the compiler.
- Dynamic types are the variable values known at execution.

Method lookup

- The compiler choses one signature (with static types)
- The runtime uses dynamic types and looks for a method compatible with the signature

What you should know

- Dynamic / static types
- equals(Object o)

A course by Stéphane Ducasse

http://stephane.ducasse.free.fr

Reusing some parts of the Pharo Mooc by

Damien Cassou, Stéphane Ducasse, Luc Fabresse

http://mooc.pharo.org