1.00
Tutorial 6

(Abstract classes, Interfaces and Pset5)

Topics

- Abstract classes
- Interfaces
- ProblemSet 5 discussion

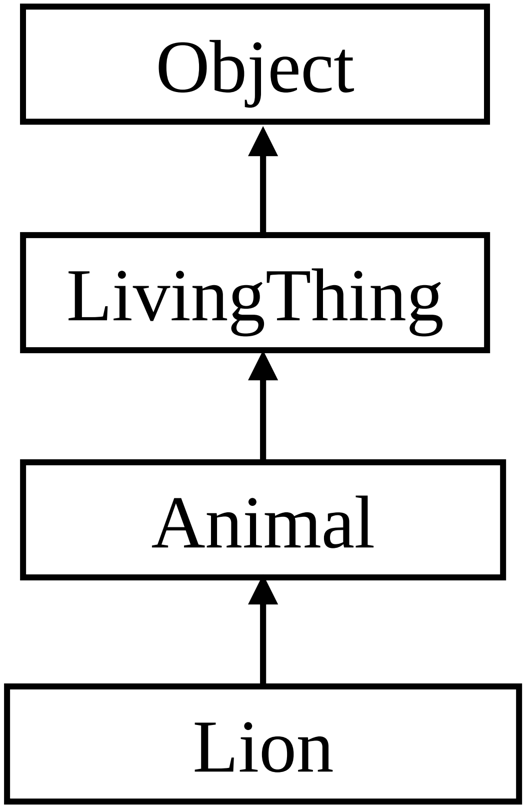
Abstract Classes

- An Abstract class cannot be instantiated
- Abstract classes can have data fields and concrete methods
- Abstract classes can also contain abstract methods
  - Any subclass must implement all of the abstract methods (provided the subclass itself is not abstract)

Abstract Class Example

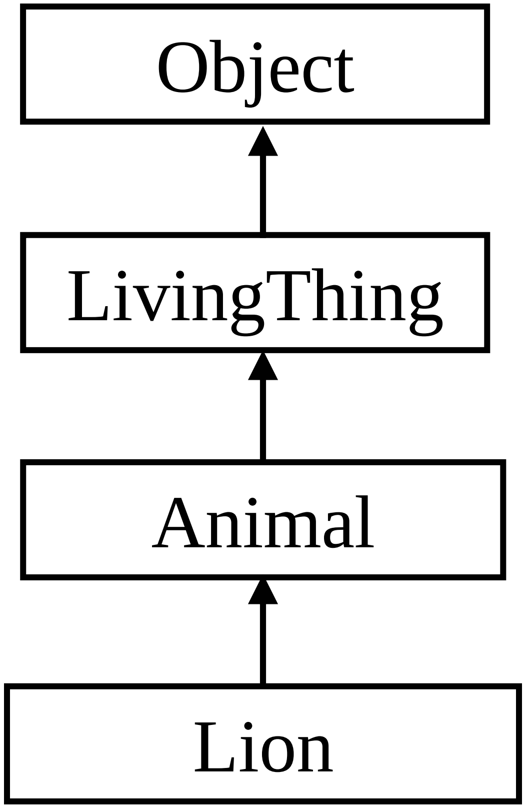
Here is modified example where now **`Animal`** extends an **abstract class `LivingThing`**

```
public abstract class
   LivingThing {
    private String habitat;
    public LivingThing(){
      habitat=“earth”;
    }

public class Animal extends
   LivingThing {
    //as before }

public class Lion extends
   Animal{
    //as before }
```



Abstract Class Questions

- Can you create an object from LivingThing ? Why ?
- Now what are the types of
  - class `Animal`
  - class `Lion`
- What fields can each of the above classes access ?



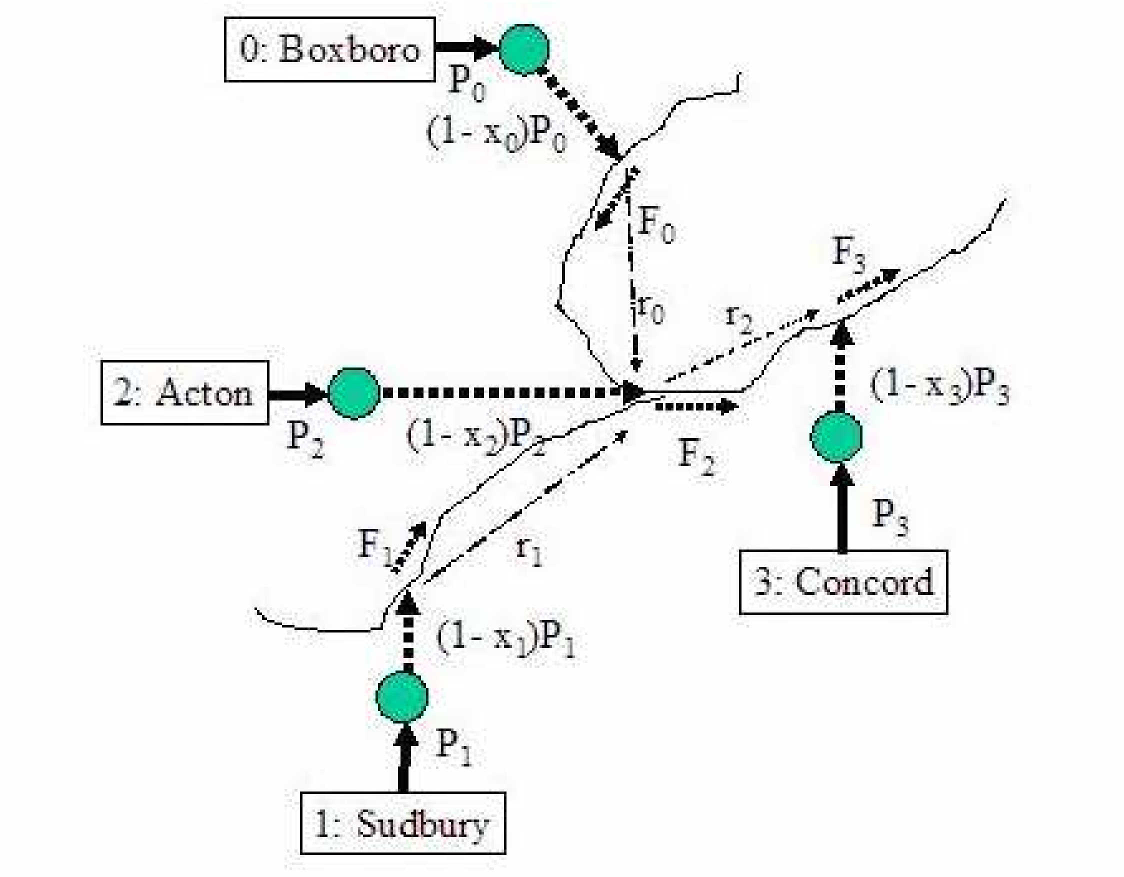
Interfaces Summary

- An interface is a collection of method declaration (and optionally, public constants).
  - All methods are abstract (but the `abstract` keyword is not used).
  - All methods are automatically public.
- An interface describes *what* its implementing classes should do
  - Ensure that some required piece of functionality is present in every implementing class.
  - Allow two totally different kinds of objects with no inheritance relationship to be handled using same code.
- Any class *implementing* a particular interface **must** define the *how*.
- Classes can implement one or more interface

Interface Exercise

- Write an interface called `Endangered`. It has one method called `getPopulation()`
- Now modify the `Lion class` so that it implements the Endangered interface
  - What additional method is required in the `Lion` class ?

PS 5: Problem Definition



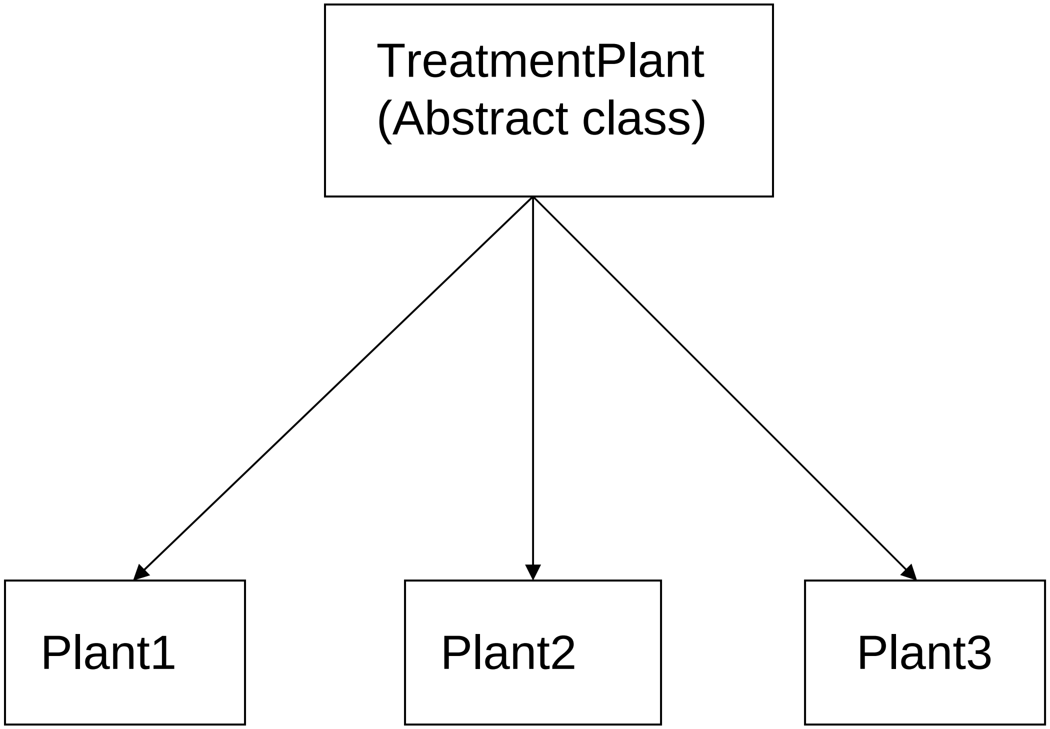
Town i:

- pollutant production rate, Pi
- pollutants discharged into river: $(1-x_i)$Pi
- Flow rate downstream: Fi
- Conc. of pollutants downstream: Ci = $[(1-x_i)$Pi + Fin Cin]/ Fi
- Fraction of pollutants removed in river: r_i

Conc. Eqn for Towns

- $C0= ((1- x0)\ P0)\ /\ F0$ (Boxboro)
- $C1= ((1- x1)\ P1)\ /\ F1$ (Sudbury)
- $C2= (F0(1-r0)C0 + F1(1-r1)C1 + (1-x2)P2)/\ F2$ (Acton)
- $C3= (F2(1-r2)C2 + (1-x3)P3)/\ F3$ (Concord)

PS 5 (2)



Plant i:

- getCost()
- getArea()
- getMaxRemoved()

Use polymorphism

PS 5 (3)

- CalculateConcentration method:
  - Need something to store data on which towns/plants are upstream of a particular town

- TreatmentPlantTest,java

  Main():

  - input plant type for each town
  - output pollution concentration downstream for each plant
  - output cost, area and maximum pollutants removed from each plant

COMPARISON OF ABSTRACT CLASSES AND INTERFACES

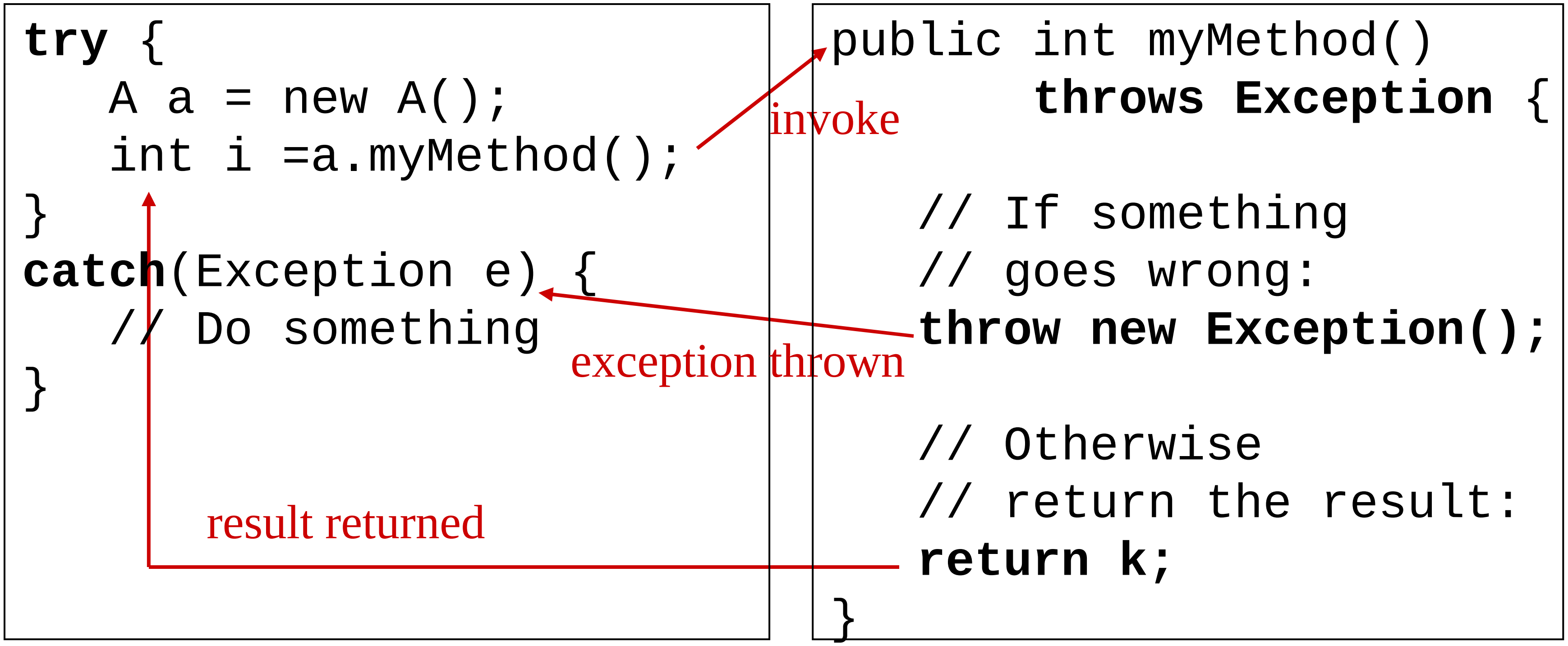
Abstract Class (A)	Interface (M)
Usually used as a base class at the top of a hierarchy (ex: Shape…)	No hierarchy implied. Can be used with disparate objects (ex: IAge, IColor…)
Other class inherit from A (keyword "extends")	Other classes implement M (keyword "implements")
A class can inherit from one abstract class only (multiple inheritance is not supported in Java)	A class can implement multiple interfaces
An abstract class can have instance variables and methods	An interface is usually a collection of method declarations only, but it also supports the declaration of constants (which are automatically final)
Methods can be private or public	All methods are automatically public
Methods can be concrete or abstract (with the keyword "abstract" used explicitly)	All methods are abstract (without actually being preceded by the abstract keyword), i.e. they have a name, return type and parameters but no implementation
Objects of A cannot be instantiated using the keyword "new" (Shape s = new Shape(); is not allowed)	Objects of M cannot be instantiated using the keyword "new" (IAge a = new IAge(); is not allowed)
A reference to an object of type A is allowed ("Shape s;" or "Shape s = new Square();" are allowed	A reference to an object of type M is allowed (" IAge a;" or " IColor c = new Wall();" are allowed)
A concrete class inheriting from A must override the abstract methods of A	A concrete class implementing M must implement ALL methods of M

Review Exception

- Used to handle malfunctions that must be processed in a different method from where they are detected.
- Programmer must work to handle the exception
- If a method can throw an exception, you can declare the type of exception in the header after the keyword throws.

Review Continued

- **try/catch & throw**



```
try {
   A a = new A();
   int i =a.myMethod();
}
catch(Exception e) {
   // Do something
}
```

```
public int myMethod()
       throws Exception {

   // If something
   // goes wrong:
   throw new Exception();

   // Otherwise
   // return the result:
   return k;
}
```

invoke

exception thrown

result returned

Exercise - Exception

- **Step 1: Complete a `static` method `factorial()`**
  - Takes non-negative integer as an argument
  - If negative number is passed, throw an `IllegalArgumentException`
  - Otherwise, calculate and return the result

Exercise - Exception

- **Step 2: Test this method with `try/catch` block**

  - Complete `catch()` block

    (how to handle the error)

  - Try `factorial(5)`

  - Try `factorial(-2)`

- **Step 3: Discuss what would happen if we didn’t use `try/catch` block**

Exercise - Exception

```
public static int factorial(int n)
              throws IllegalArgumentException {
  if (n < 0) { throw new IllegalArgumentException(); }
  int result = 1;
  for (int i = n; i > 0; i--) { result *= i; }
  return result;
}

public static void main(String[] args) {
  try {
      int a = factorial(5);
      int b = factorial(-2);
  }
  catch (IllegalArgumentException ex) {
      System.out.println("Invalid input");
  }
}
```