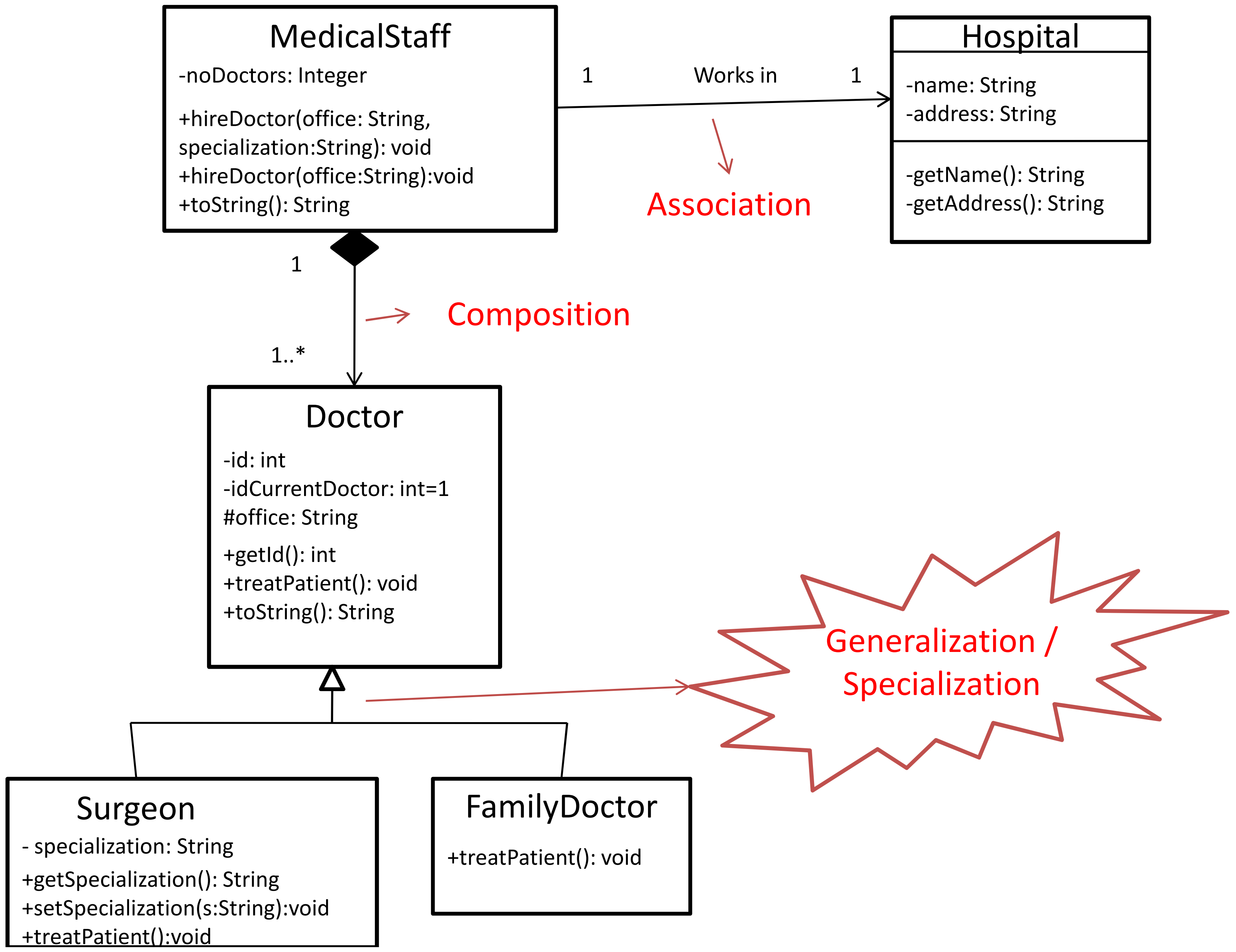
Inheritance. Static fields and methods. Static and dynamic polymorphism

Maria-Iuliana Dascalu
mariaiuliana@gmail.com

FILS, UPB, 2011
Programming Languages

Relationships between classes

Manage Hospital System



Inheritance

- Used for generalization/specialization relationship ("is-a" relationship)

- A subclass inherits/extends the fields and methods (which are <u>not private</u>) of a superclass:
  - Adds new fields/methods;
  - Modifies (<u>overrides</u>) fields/methods;

- Keyword in Java for denoting inheritance: *extends*

- In Java, a class can inherit only one superclass

Example

```
public class Surgeon extends Doctor {}
```

subclass superclass

Exercise: Open the “ManageHospital” project and make the FamilyDoctor class a subclass of the Doctor class..

Accessibility Levels

- Public members are inherited: the subclass “sees” the public members of the superclass

- Private members are not inherited: to access the private members of the superclass in subclass, we have to create getters

- Protected members are inherited: the subclass “sees” the public members of the superclass; for the other classes, protected members act as private members

Example

```
public class Doctor{
    //subclasses Surgeon and FamilyDoctor can’t access
    the id, so we create a getter
    private int id;
    // the office location is “seen” by the two subclasses
    protected String office;
    public int getId(){
            return id;
    }
}
```

Constructors

- The constructor of the subclass always calls a constructor of the superclass, explicitly or implicitly

- For avoiding compilation errors, we have several possibilities:
  - In the subclass constructor, we have to call explicitly one of the superclass constructors (using the keyword *super*)
  - Insert the constructor without any parameters defined in the superclass (because it is called implicitly by the subclass constructor)

Example

```
public Doctor(String office)
{
      //each doctor has to have an office
      this.office = office;
      //each doctor has to have a unique id
      //we use a static field to help us ensuring the //uniqueness of each object id
      within the Doctor class
      this.id=idCurrentDoctor;
      idCurrentDoctor++;
      //it makes NO SENSE to use "this" for a static member
}
public Surgeon(String office)
{
      // calls the explicit constructor of the superclass
      super(office);
}
```

Exercise: Implement the constructor of the class FamilyDoctor.

Using Subclass Objects

```
Surgeon d1=new Surgeon(“AB01”);
// notice that the toString method is not defined
// in the subclass Surgeon, it is inherited from
// the superclass Doctor
System.out.println(d1);
Doctor d2=new Surgeon(“AB06”);
System.out.println(d2);
```

Exercise: Run the application and notice the uniqueness of each doctor id!

Using Subclass Objects

```
Surgeon d1=new Surgeon(“AB01”);
// notice that the toString method is not defined
// in the subclass Surgeon, it is inherited from
// the superclass Doctor
System.out.println(d1);
Doctor d2=new Surgeon(“AB06”);
System.out.println(d2);
```

Exercise: Run the application and notice the uniqueness of each doctor id!

Casting & *instanceof*

```
Doctor d3=new Surgeon(“AB09”);
D3.setSpecialization(“cardiology”);
```

Exercise: Compile the application! What happens?

Casting & *instanceof*

=> compilation error

The d3 reference variable is of superclass type.

The object to which it refers is of subclass type.

The compiler does not know that!

Solution: use *instanceof*

The *instanceof* keyword can be used to test if an object is of a specified type:

if (objectReference **instanceof** type)

Exercise: Check if d3 is of type Surgeon and, in this case, set the specialization using the setSpecialization method. Do this in the main method of the ManageHospital class.

You will also need an explicit cast between the subclass and the superclass!

Polymorphism

- A characteristics of an entity to have different significations, depending on the context.
- Examples of polymorphism:
  – overloaded constructors

  Exercise: Write another constructor for the class Surgeon with 2 parameters: String office, String specialization.

  – overridden methods

  Exercise: Analyze the classes treatPatient() from Doctor, Surgeon and FamilyDoctor! The toString method is overridden: it is defined in the Object class and redefined in our classes.

Overloading *vs.* Overriding

- Overloading :
  - takes place when we have more methods with the same name in the same class, but different parameters;
  - takes place during compilation => static polymorphism.
- Overriding:
  - takes place when we redefine superclass methods within subclasses, keeping the same name, but changing the implementation;
  - takes place at runtime => dynamic polymorphism.

Exercises

- Take a look at class MedicalStaff.
- Created 2 overloaded hireDoctor methods.
- Test the MedicalStaff and Hospital classes:
  - Create an object of type MedicalStaff;
  - Change the hospital field of it. What Happens?

Final Keyword

- Attaching the *final* keyword to a class => that class can not be inherited
- Attaching the *final* keyword to a method=> that method can not be inherited
- Attaching the *final* keyword to a field=> that field can not be changed after instantiation.

Abstract classes

- They have no direct instances, but they do have subclasses with concrete instances.
- They must have at least one abstract method (a method in which no algorithm is defined and must be overridden in the subclasses).

Example:

```
public abstract class AAA{
            ………..
            public abstract void aaa (); // no body
}
```

- You can not call the constructor of an abstract class!!!

Exercise: Make the treatPerson from the Doctor class an abstract method!

Exercises

- Create another subclass of the class Doctor. Named it ResidentDoctor and write some appropriate fields/methods for it (at least, override the method treatPatient!!!).
- Test all your classes: add 5 doctors to your hospital and establish their activities, then display all your information! Your result should look like this:

[5] doctors are hired at the [General] hospital. Their activities are:

Doctor [1] having the specialization [cardiology] [operates patients].

Doctor [2] [gives prescriptions].

Doctor [3] having the specialization [neurology] [operates patients].

Doctor [4] [gives prescriptions].

Doctor [5] [makes what a resident does].

- Implement the toString method from class MedicalStaff or whatever methods you need to obtain the above result.