File Edit View Help

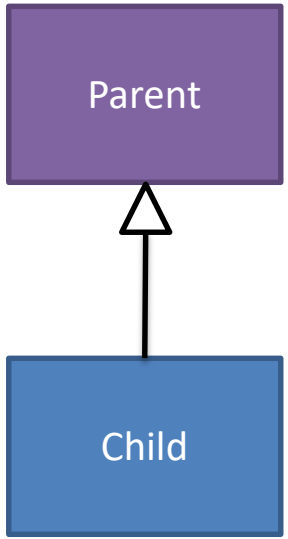
File Edit View Help

Inheritance and Polymorphism Part 01

Inheritance

- Inheritance allows Data and Methods to be *inherited / absorbed* from one class into another
- In Java, this occurs between two classes
  - Subclass (Child): The class inheriting from another
  - Superclass (Parent): The class that is being inherited
- This is great for *extending* the properties and functionality of one class into another
  - The subclass becomes a more specific version of the superclass
  - The superclass is a more general version of the subclass
  - Creates an “is a” relationship

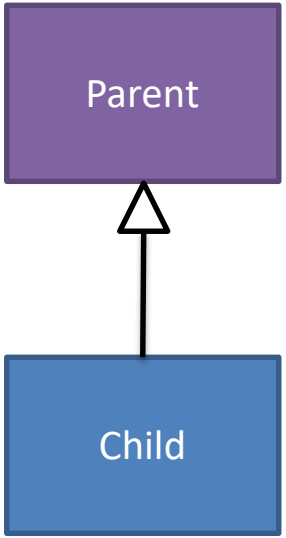
Inheritance Concept



Inheritance

- Private properties and methods are not inherited from the superclass
- Instance variables from the superclass must be accessed through their accessors
- Instance variables from the superclass must be modified through their mutators

Inheritance Concept



Inheritance

- The reserved word “extends” is used to inherit from one class into another
- The reserved word “super” is used to call methods from the superclass
  - To call a superclass’ constructor it should use “super()” or “super(<<parameters>>)”
  - To call any other superclass’ methods user “super.<<method id>>(<<parameters>>)”
  - To access properties in the superclass use “super.<<accessor>>”
  - To modify properties in the superclass use “super.<<mutator>>”

Inheriting a Class Syntax

```
public class <<subclass id>> extends <<superclass id>>
{
      //Body of the class
}
```

Example

```
public class Student extends Person
{

}
```

Inheritance

- In Java, a subclass does not inherit constructors from the superclass
- The reserved word “super” should used to call the superclass’ constructor
  - Generally, it should be the first call in a subclass’ constructor
  - “super()” is the call to the superclass’ default constructor
  - “super(<<SC’s params>>)” is the call to the superclass’ (SC’s) parameterized constructor
- The subclass’ parameterized constructor should also include the superclass’ properties as parameters

Inheritance Constructor Syntax

```
public <<subclass id>>()//Default Constructor
{
      super();//Call to super class’ default constructor
      …
}
```

Example

```
public Student()
{
      super();//Call to Person’s default constructor
      …
}
```

Inheritance

- In Java, a subclass does not inherit constructors from the superclass
- The reserved word “super” should used to call the superclass’ constructor
  - Generally, it should be the first call in a subclass’ constructor
  - “super()” is the call to the superclass’ default constructor
  - “super(<<SC’s params>>)” is the call to the superclass’ (SC’s) parameterized constructor
- The subclass’ parameterized constructor should also include the superclass’ properties as parameters

Inheritance Constructor Syntax

```
public <<subclass id>>(<<SC’s params>>, <<subclass’ params>>)
{
        //Call to super class’ default constructor
        super(<<SC’s params>>);
        …
}
```

Example

```
public Student(String aN, int anID)
{
        super(aN);//Call to Person’s param constructor
        …
}
```

Overriding Methods

- Overriding a method is when a subclass has the same *signature* or *definition* of a method in the superclass
  - Different from Overloading Methods
- A method from the superclass can be called from the subclass by using "super.<<method id>>"
- This can be useful when overriding the ".equals()" and ".toString()" method
- The reserved word "final" can be used so a method or class CANNOT be overridden or extended

Inheritance "toString()" Syntax

```
public String toString()
{
      return super.toString() + <<other properties>>
}
```

Example

```
//Student's toString method
public String toString()
{
      return super.toString() + "ID: "+this.id;
}
//super.toString() is a call to Person's toString
//method
```

Overriding Methods

- Overriding a method is when a subclass has the same *signature* or *definition* of a method in the superclass
  - Different from Overloading Methods
- A method from the superclass can be called from the subclass by using "super.<<method id>>"
- This can be useful when overriding the ".equals()" and ".toString()" method
- The reserved word "final" can be used so a method or class CANNOT be overridden or extended

Inheritance "equals()" Syntax

```
public boolean equals(<<AI>>)
{
      return super.equals(<<AI>>) &&
                  <<property checks>>
}
```

Example

```
//Student's equals method
public boolean equals(Student aS)
{
      return aS != null && super.equals(aS) &&
            this.id == aS.getID();
}
//super.equals() is a call to Person's equals //method
```

File Edit View Help

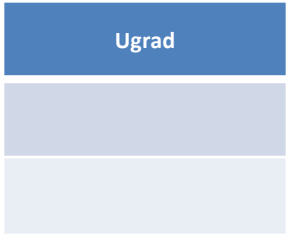
Example

Personnel System

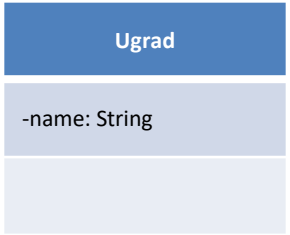
- Problem: We must keep track of important information about people at a University
- Different types include:
  - Undergraduate Students
  - Graduate Students
  - Faculty
  - Staff
- Undergraduate Information
  - Name
  - Student ID
  - Level
- Graduate Information
  - Name
  - Student ID
  - Advisor's Name
- Faculty Information
  - Name
  - Salary
  - Courses
- Staff Information
  - Name
  - Salary
  - Supervisor
- Should be able to:
  - Add new people
  - Remove people
  - View all people in the system
- Clear and Easy-to-Use Frontend

File Edit View Help

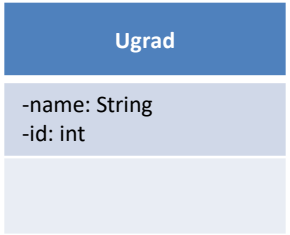
File Edit View Help



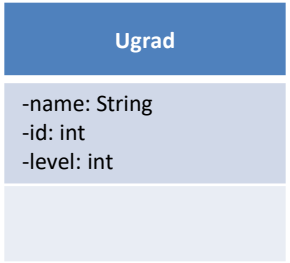
File Edit View Help



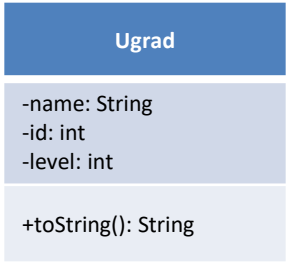
File Edit View Help



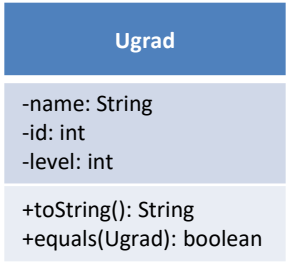
File Edit View Help



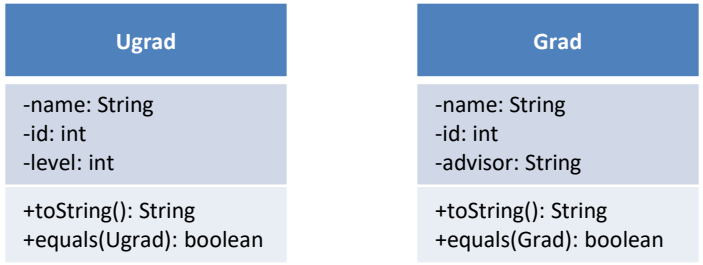
File Edit View Help



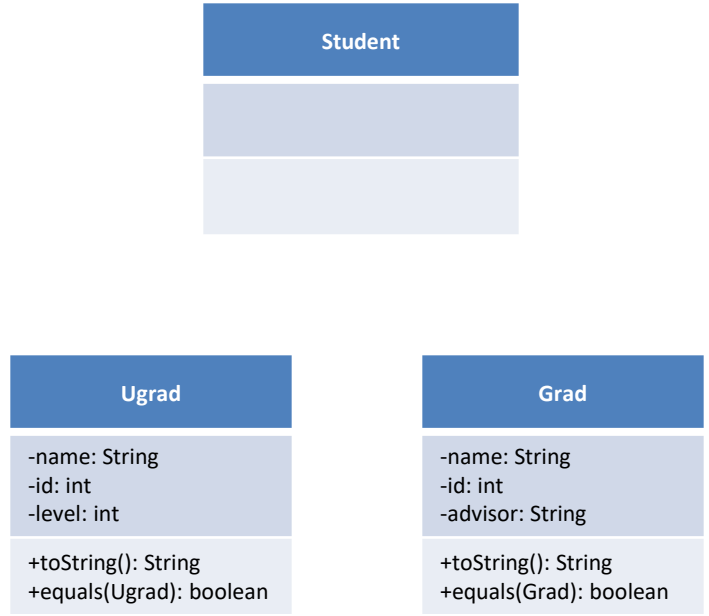
File Edit View Help

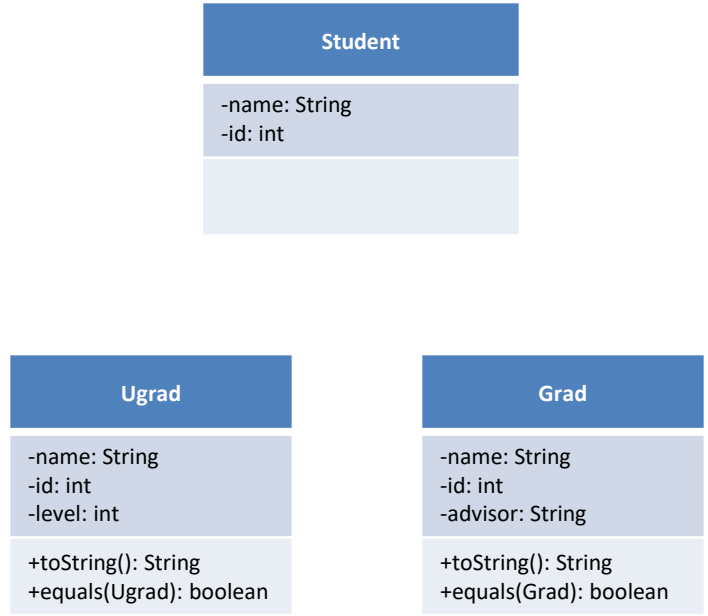


File Edit View Help

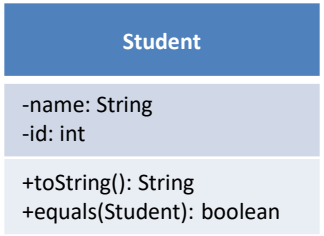


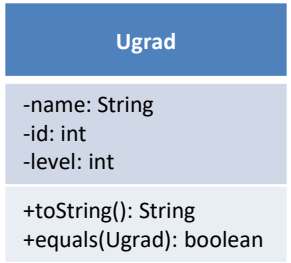
File Edit View Help

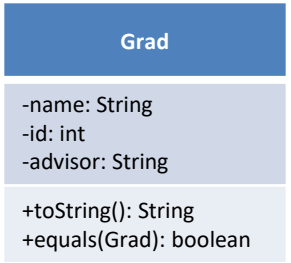


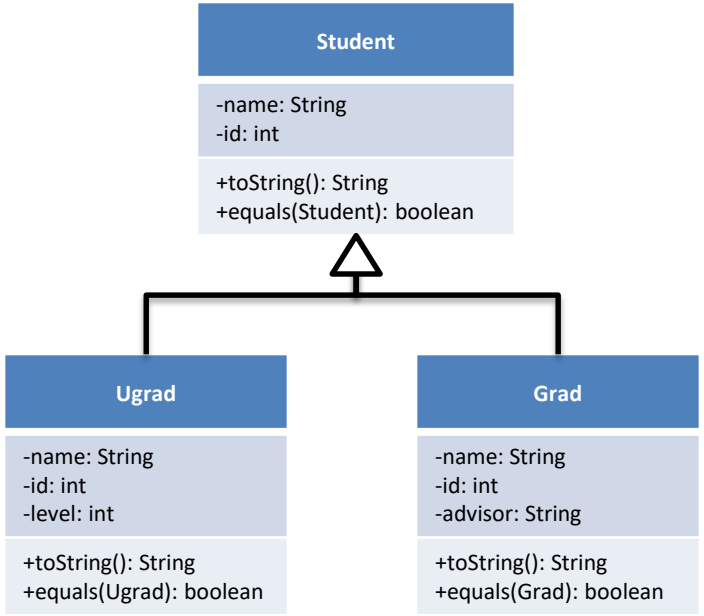


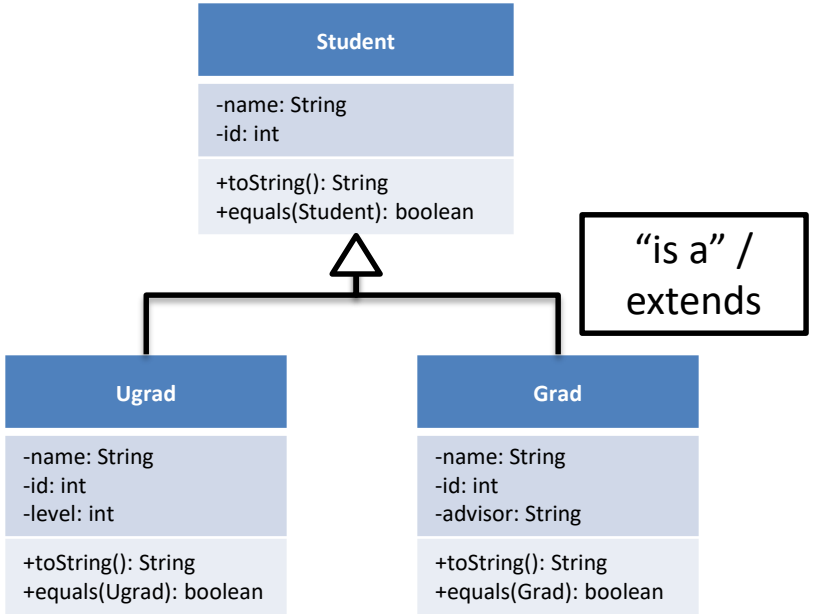
File Edit View Help

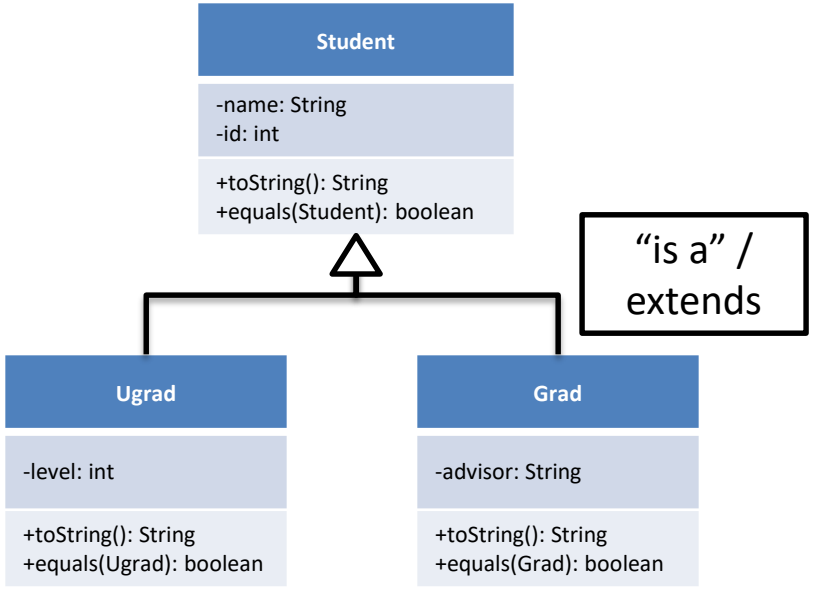




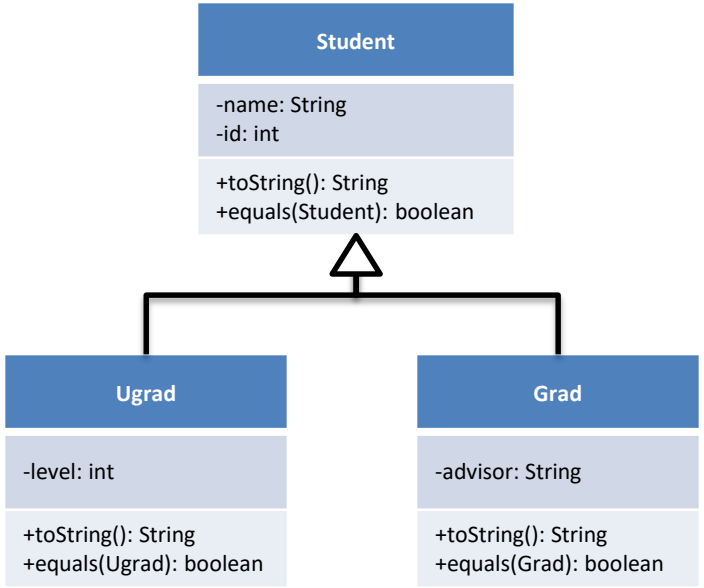


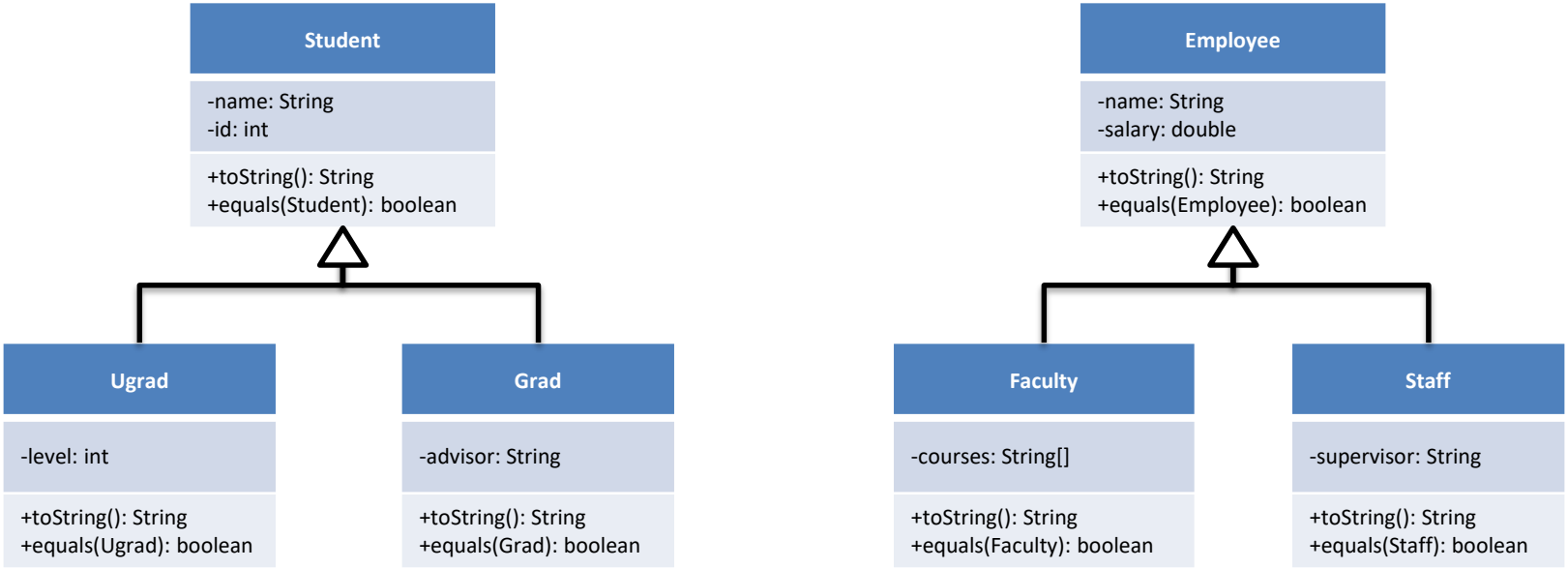


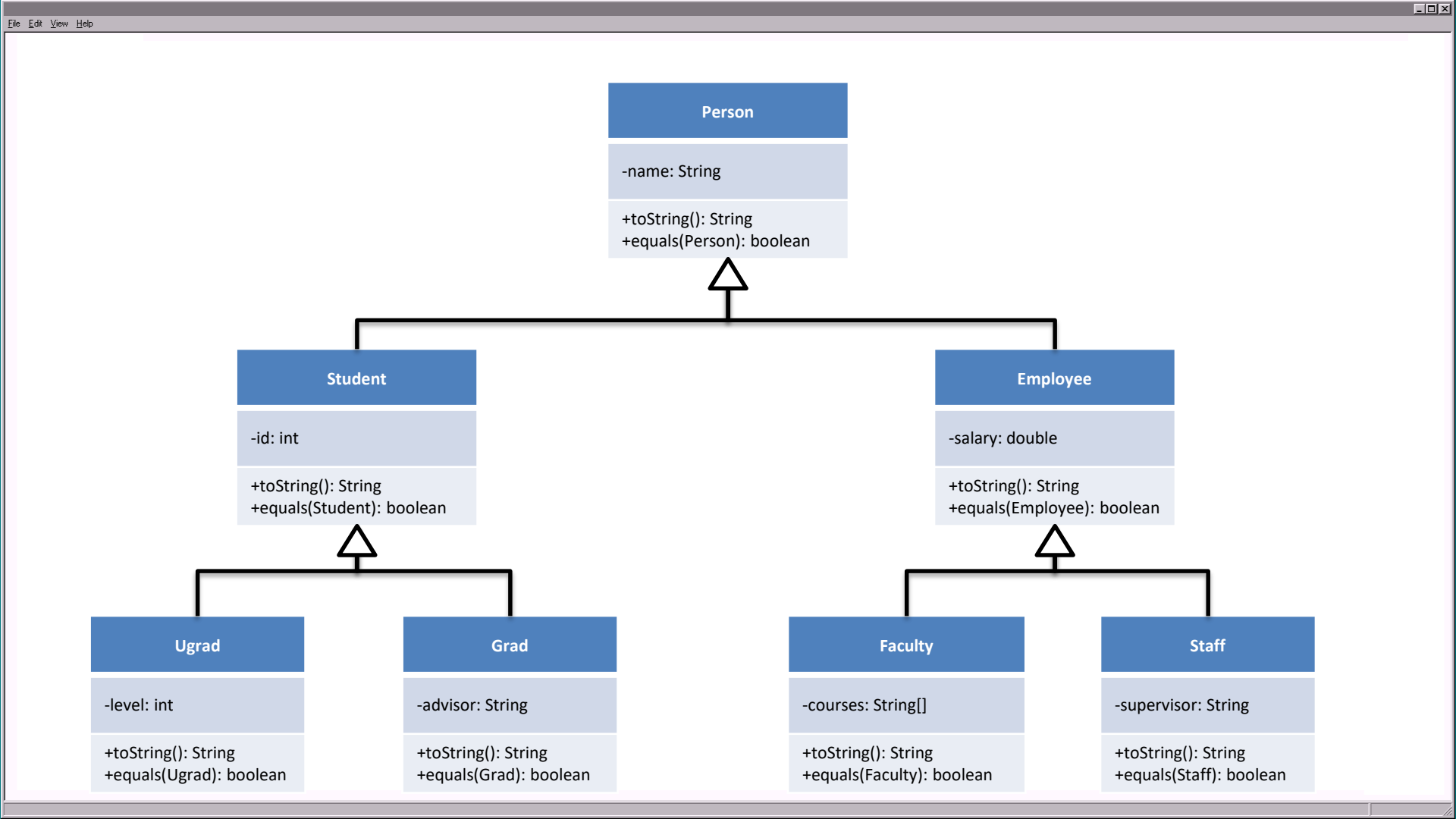


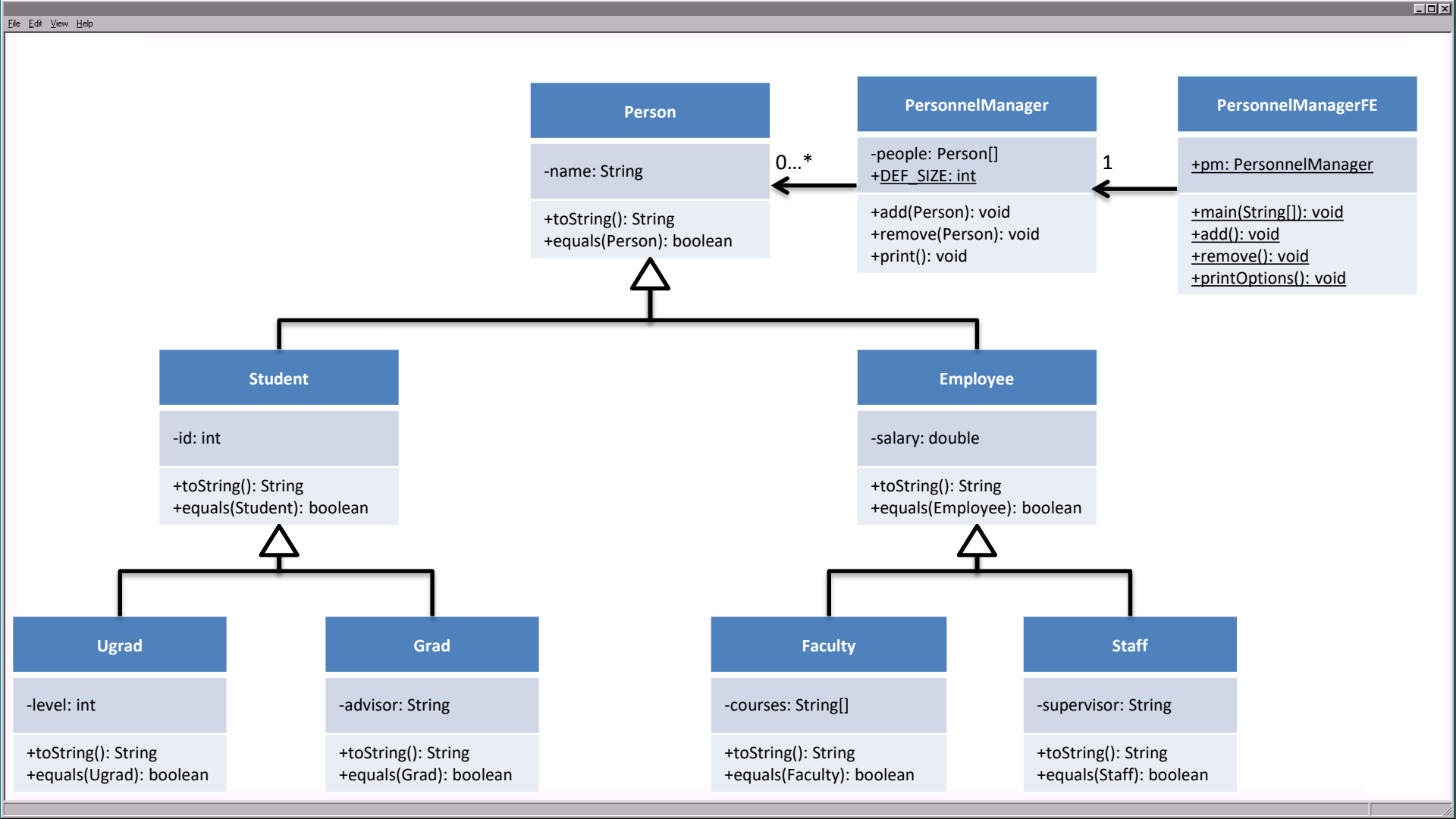


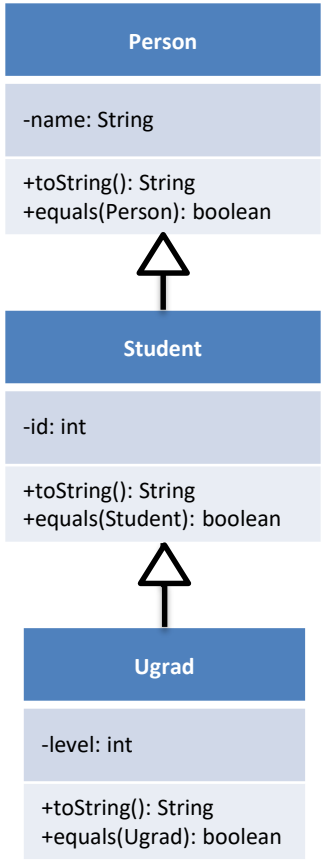
File Edit View Help

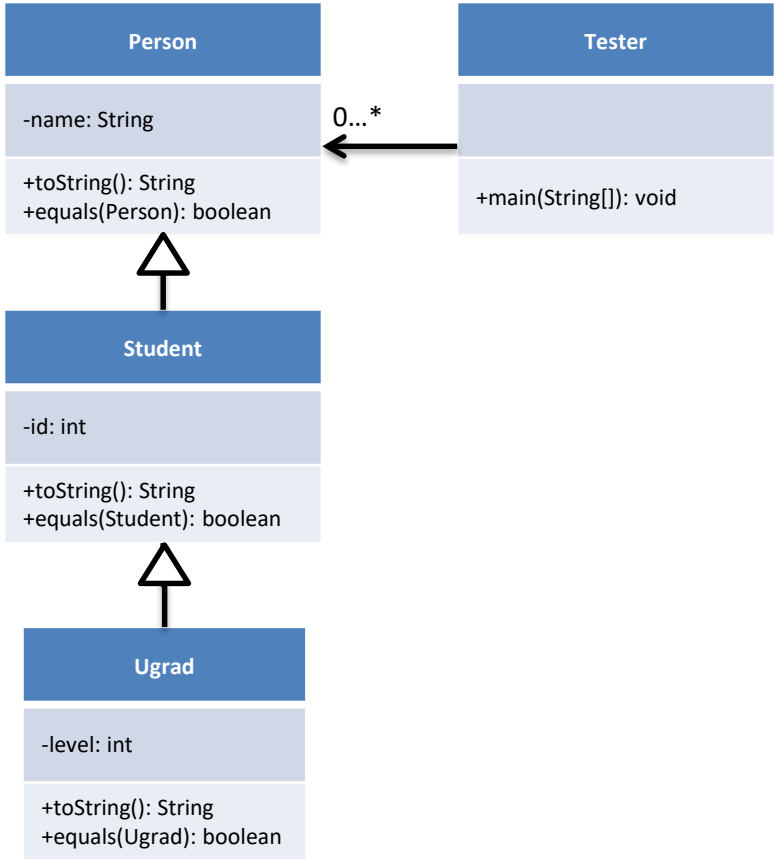





File Edit View Help
Person
-name: String
+toString(): String
+equals(Person): boolean
Student
-id: int
+toString(): String
+equals(Student): boolean
Employee
-salary: double
+toString(): String
+equals(Employee): boolean
Ugrad
-level: int
+toString(): String
+equals(Ugrad): boolean
Grad
-advisor: String
+toString(): String
+equals(Grad): boolean
Faculty
-courses: String[]
+toString(): String
+equals(Faculty): boolean
Staff
-supervisor: String
+toString(): String
+equals(Staff): boolean


File Edit View Help
Person
-name: String
+toString(): String
+equals(Person): boolean
0...*
PersonnelManager
-people: Person[]
+DEF_SIZE: int
+add(Person): void
+remove(Person): void
+print(): void
1
PersonnelManagerFE
+pm: PersonnelManager
+main(String[]): void
+add(): void
+remove(): void
+printOptions(): void
Student
-id: int
+toString(): String
+equals(Student): boolean
Employee
-salary: double
+toString(): String
+equals(Employee): boolean
Ugrad
-level: int
+toString(): String
+equals(Ugrad): boolean
Grad
-advisor: String
+toString(): String
+equals(Grad): boolean
Faculty
-courses: String[]
+toString(): String
+equals(Faculty): boolean
Staff
-supervisor: String
+toString(): String
+equals(Staff): boolean


Person
-name: String
+toString(): String
+equals(Person): boolean
Student
-id: int
+toString(): String
+equals(Student): boolean
Ugrad
-level: int
+toString(): String
+equals(Ugrad): boolean



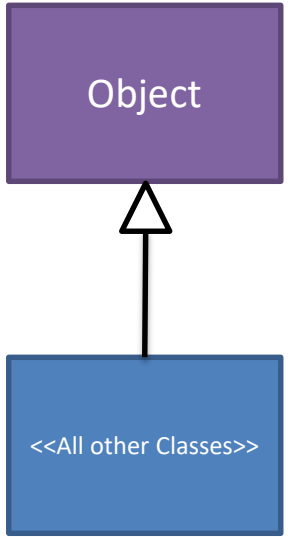
File Edit View Help

Example

Inheritance

- Every class in Java extends from the type “Object”
- Many methods are already assumed to be in each class
  - toString()
  - equals()
- We override the “toString()” and “equals()” methods to define our own specific version
  - The “toString()” method is called every time a class’ id is an argument for “System.out.println()”

Inheritance Concept



Polymorphism

- "One becomes many"
- A superclass can be extended or implemented in many different ways
- A change to a superclass is reflected across all subclasses
- Allows substitution of one class for another as long as the class *is an* extension
  - This is how the "equals()" methods works for different types
- Made possible by *dynamic binding* aka *late binding*

Polymorphism Concept