File Edit View Help

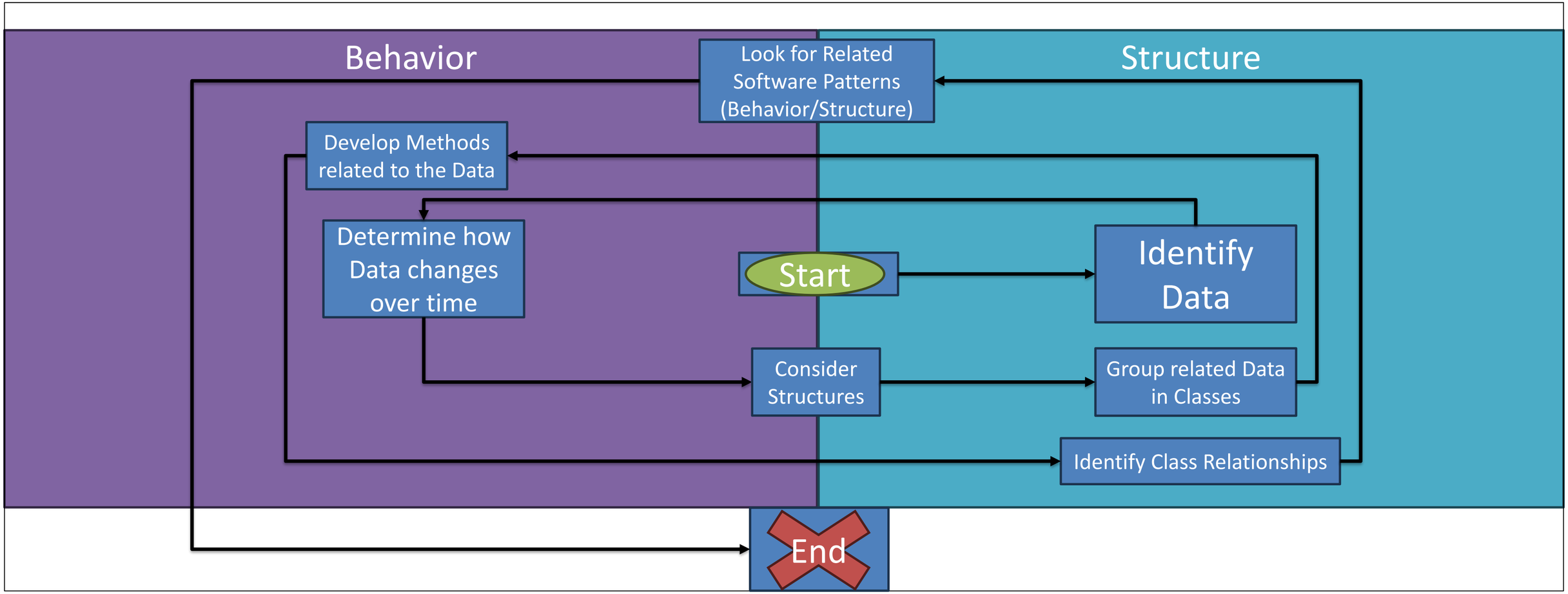
File Edit View Help

Programming Review

Part 04

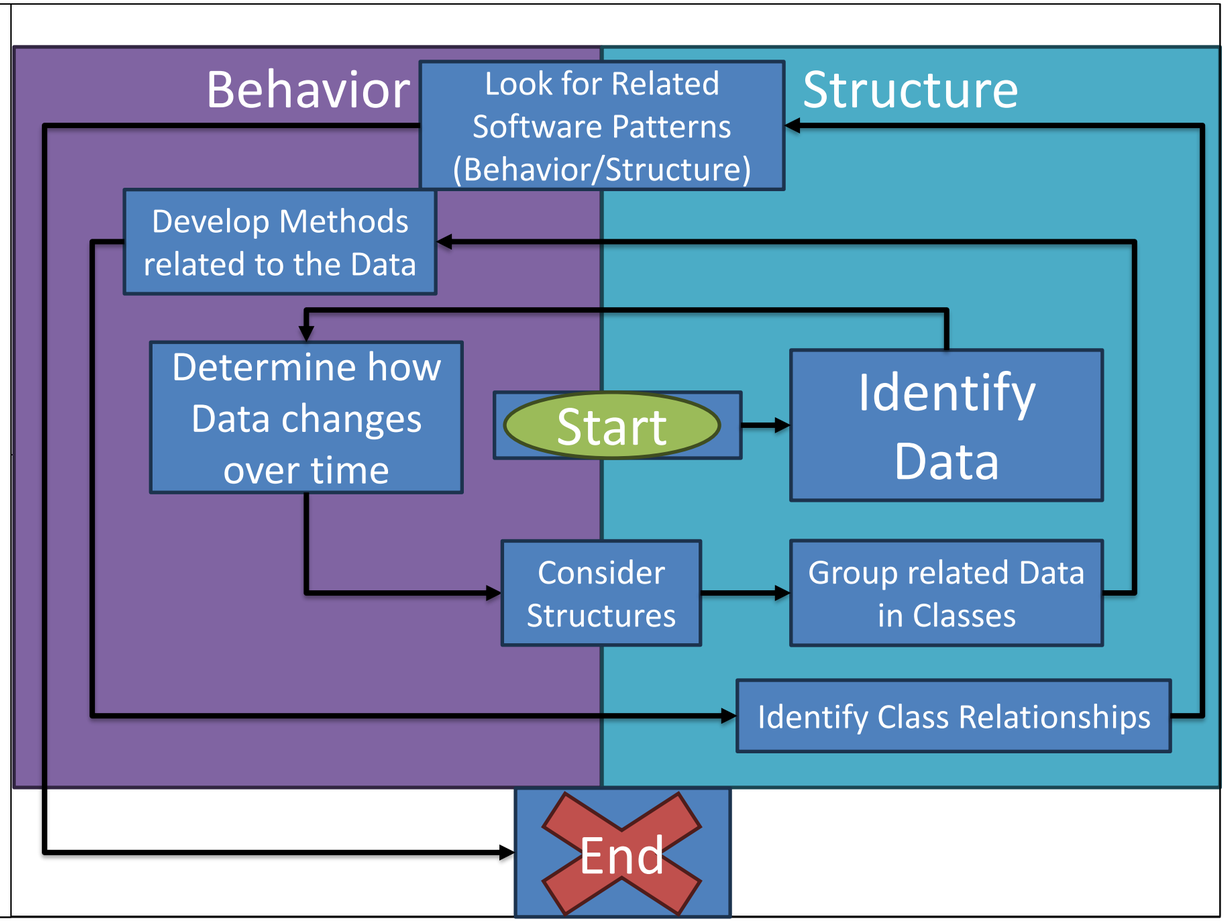
File Edit View Help

Problem Solving



Problem Solving

1. Identify your Data.
2. Determine how the data changes over time.
3. Consider structures for both behavior and data.
4. Group together (encapsulate) related information into Classes of Objects.
5. Develop functionality / methodologies that relates to the behavior of your Objects.
6. Further identify relationships between the Classes and optimize the structure.
7. Determine if there exists software patterns that may assist.

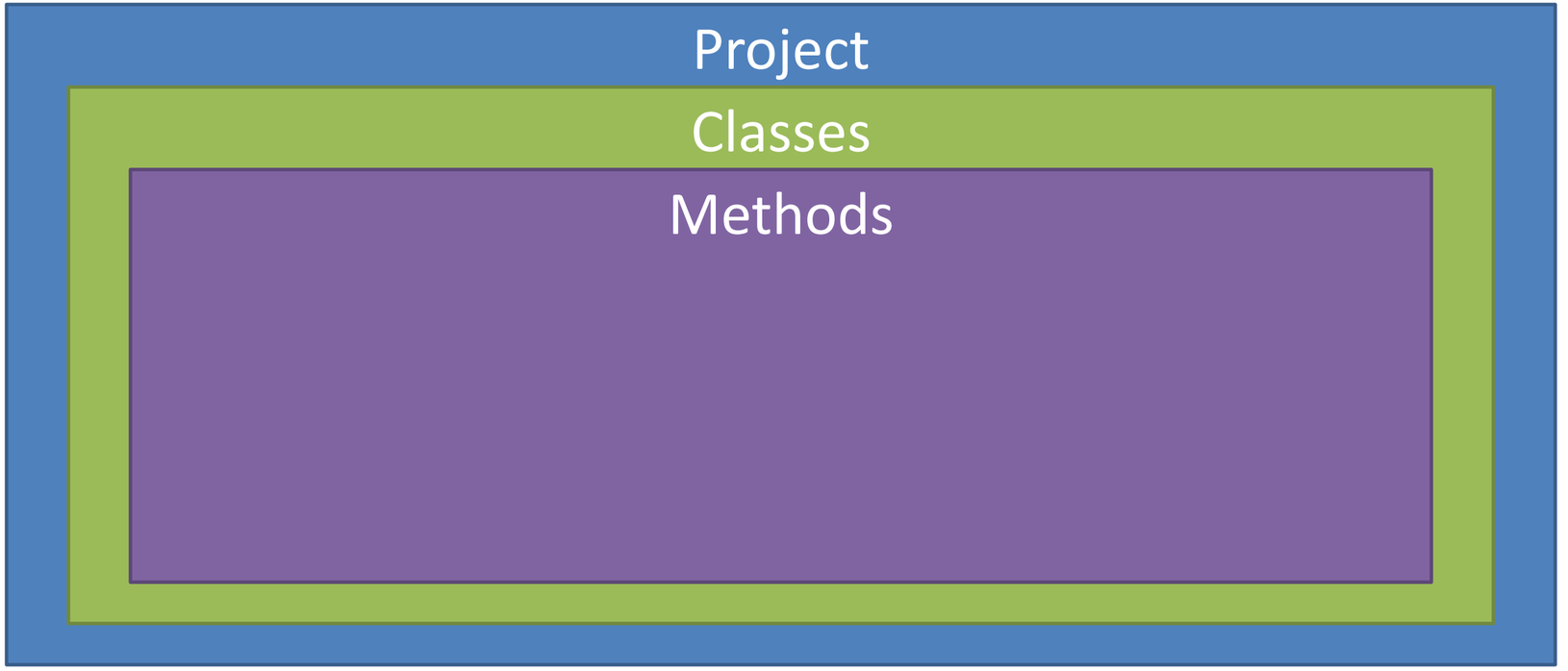


File Edit View Help

More Code Organization

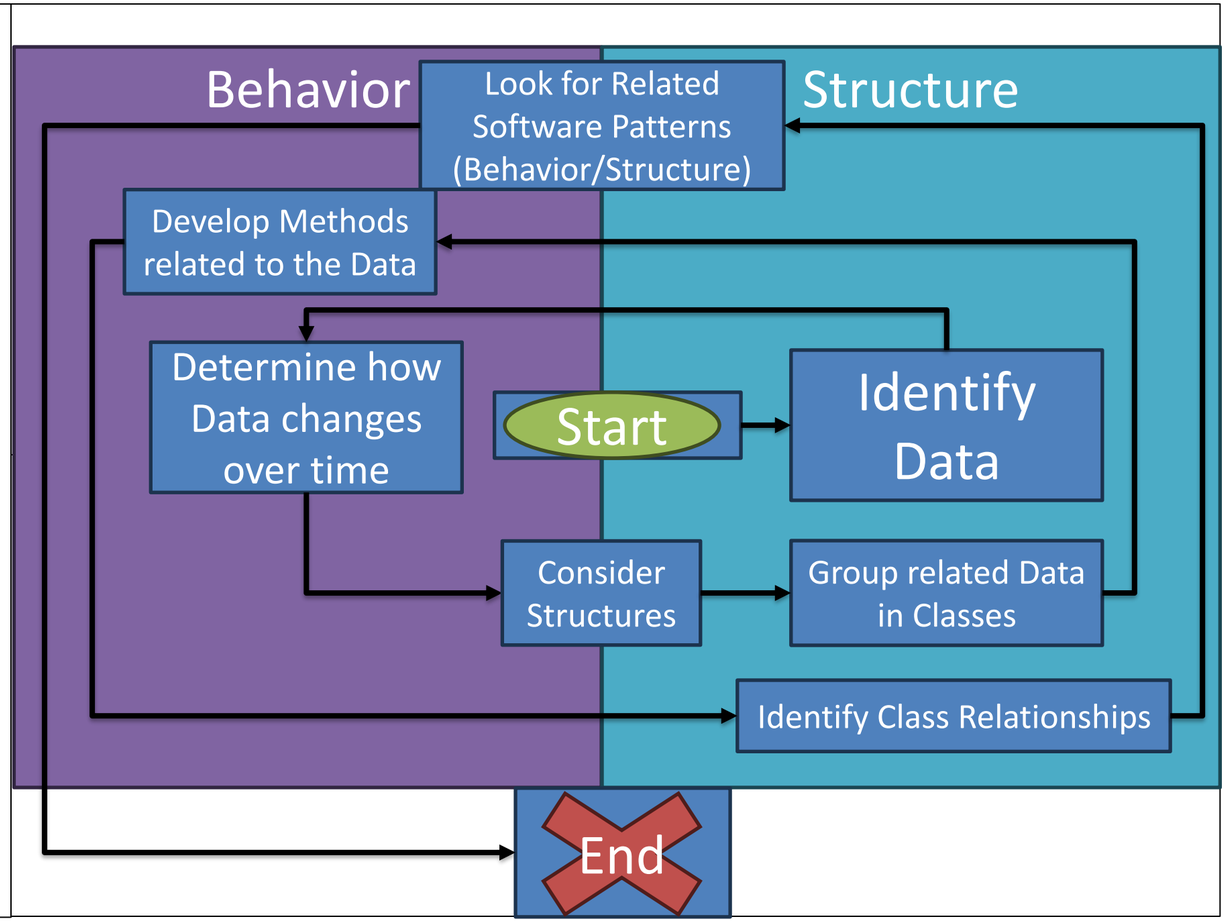
More Object-Oriented Programming

File Edit View Help



Problem Solving

6. Further identify relationships between the Classes and determine where design optimizations may occur.
   - **Association** = “Has A”
     - Contains some *Thing* else
   - **Inheritance** = “Is A”
     - Extends some *Thing* else
   - **Polymorphism** = “Does an Action”
     - Implements an *Action* in some way
   - UML Class Diagrams = Structure
   - UML Sequence Diagrams = Behavior

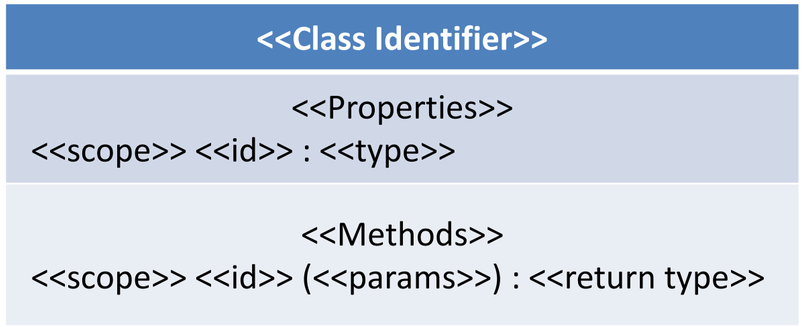


File Edit View Help

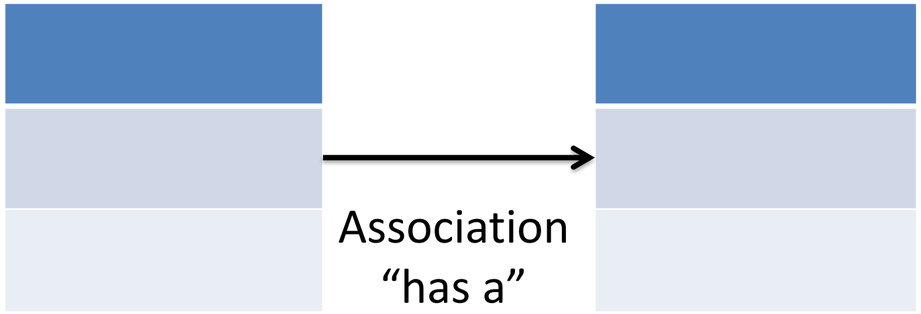
Design Diagram

- UML Class Diagram
  - –Boxes => Structures (like Classes)
  - –Arrows => Relationships between structures
- Classes
  - –Name of the class
  - –Properties
  - –Methods
  - –“+” / “-” means scope is public or private
- Arrows
  - –Stick arrow is the Association or “has a”
    - Numeric values indicate the number of instances
- Static variables and method are <u>underlined</u>
  - –Constants are all UPPER CASE

UML Class Diagram

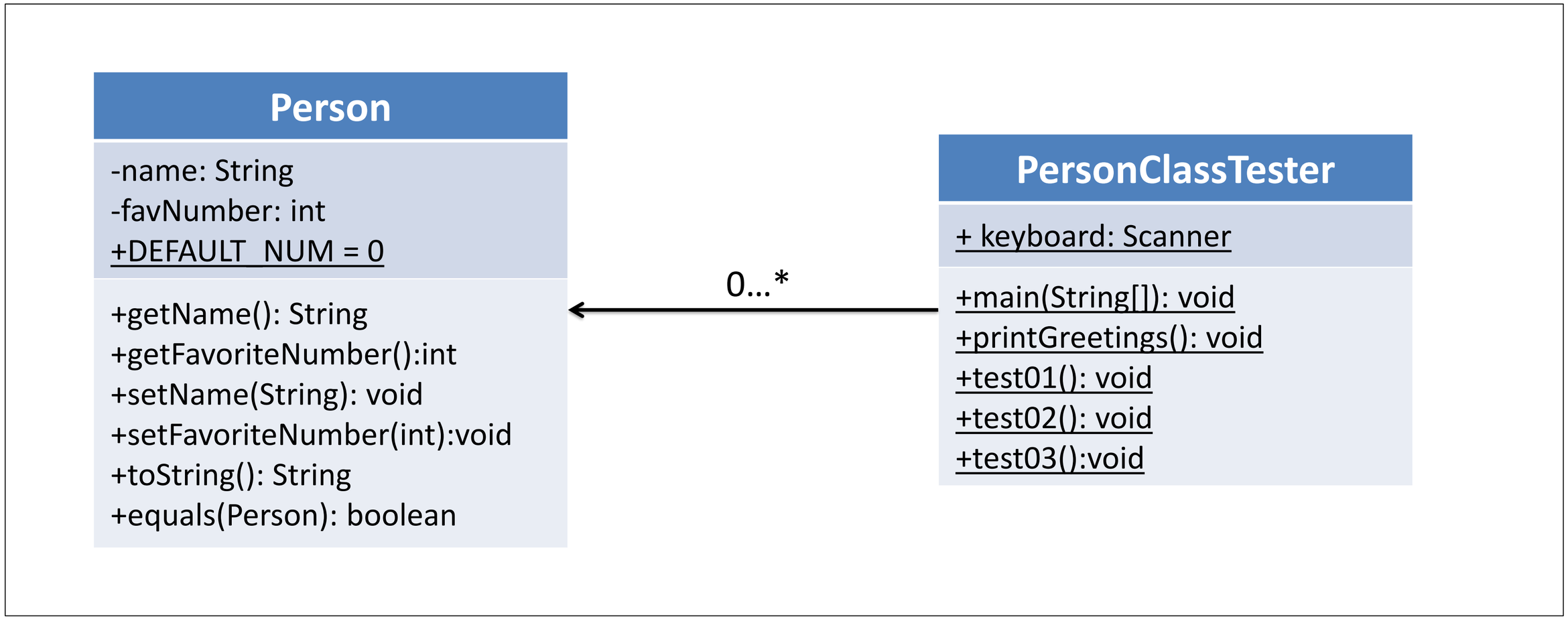


UML Class Diagram Arrows



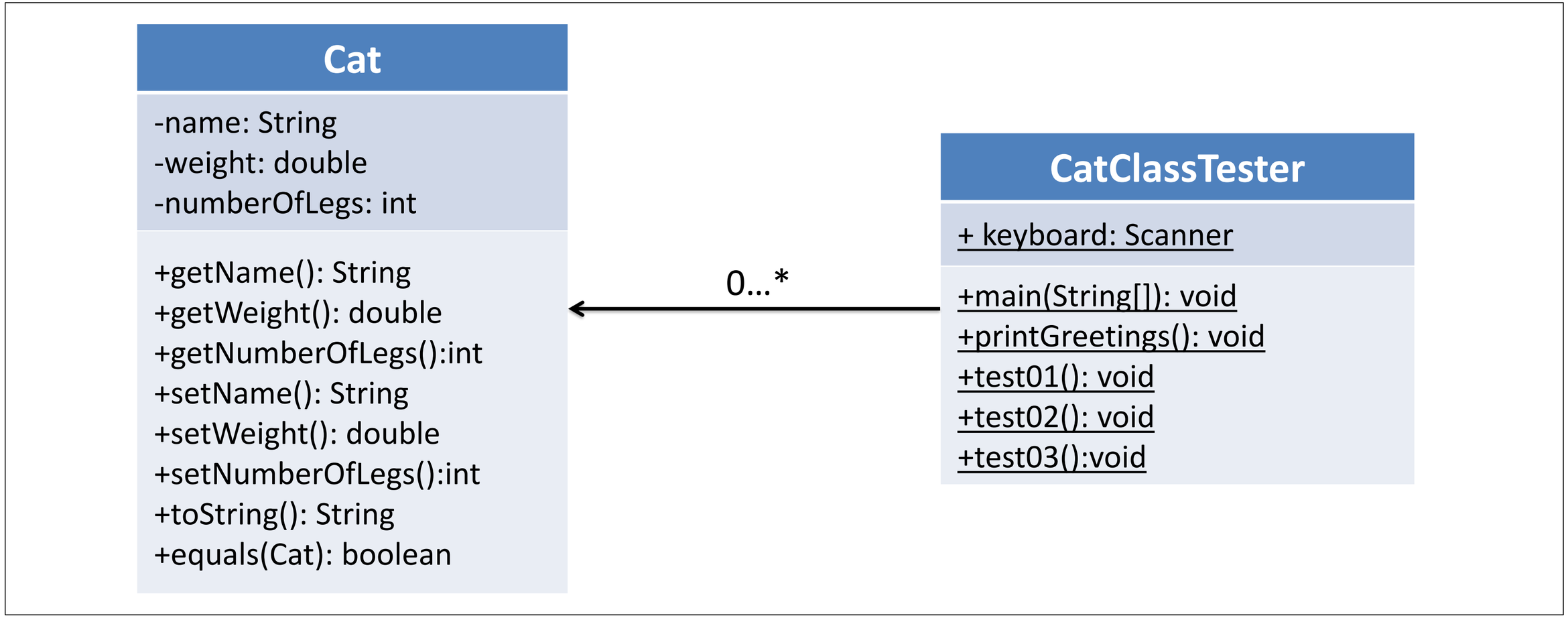
File Edit View Help

Design Diagram



File Edit View Help

Design Diagram



File Edit View Help

Inheritance

- Establishes a relationship between two classes where properties and methods are absorbed and extended from one class into another.
- Similar to Biological Inheritance
- Used to create a more *specific* version of a given class
  - Creates an “is a” relationship
  - “This *is a* that with more details”

Syntax

```
//Class def
public class <<class identifier>> extends <<other class>>
{
        …
        public <<class identifier>>()
        {
                super();//Call other class’ default constructor
        }
}
```

Example

```
public class Employee extends Person
{
     protected int id;
     public Employee()
     {
          super();//Call to Person’s def constr
          this.id = 0;
     }
     …
     public boolean equals(Employee e)
     {
          return e != null && super.equals(e) && this.id == e.getID();
     }
}
```

Inheritance

- Reserved word "**extends**" is used to establish this relationship in the Class definition
- The scope operator "**protected**" is preferred for instance variables.
  - "private" is never directly accessible outside of the class it was declared.
  - "protected" provides "public" access for inherited classes, and "private" access for everything else.
  - A scope in between "private" and "public"
- The reserved word "**super**" can be used to access methods and constructors from the parent class
  - super() is used to call the parent's constructors
  - super.<<method>> is used to call the parent's method

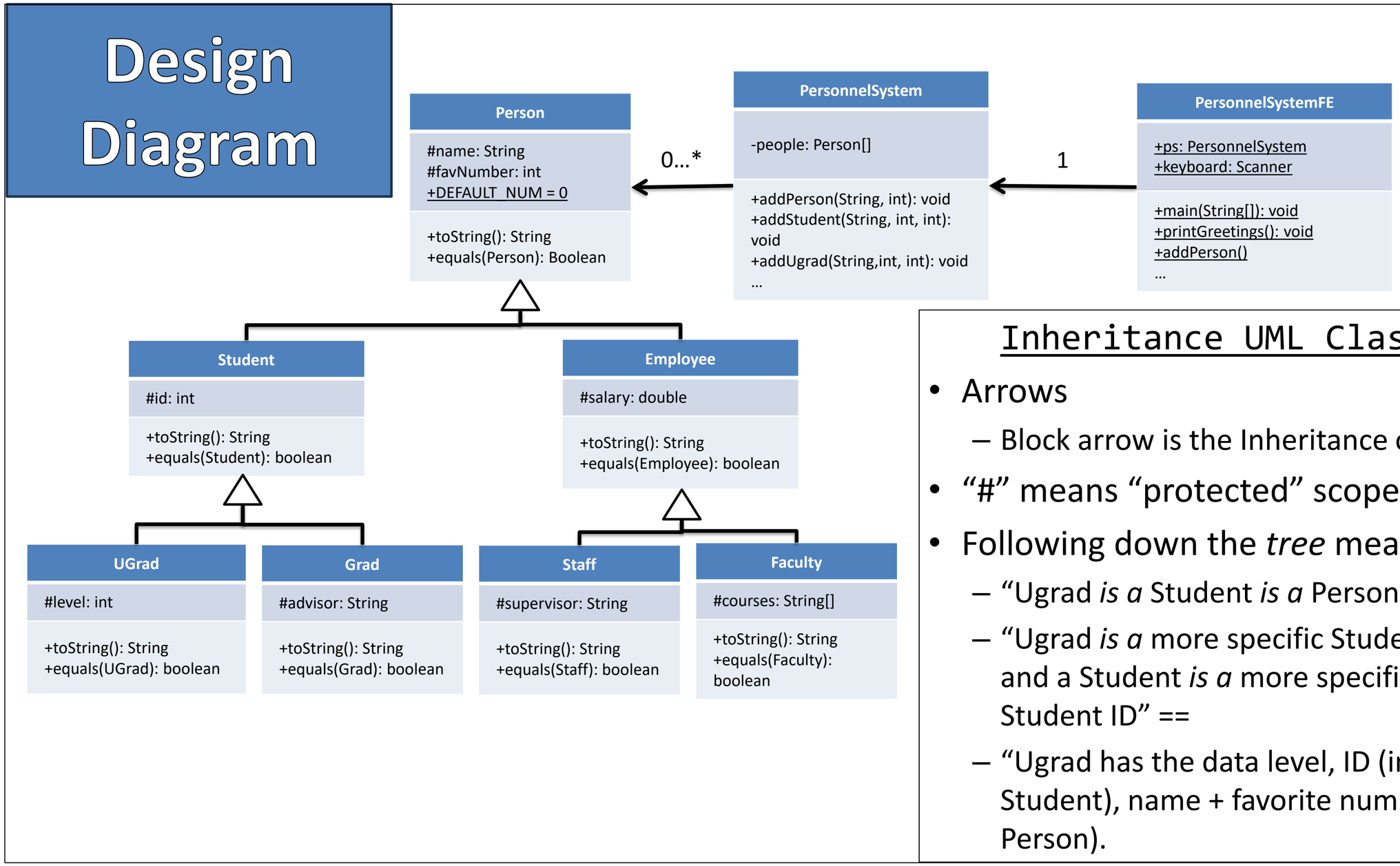
Syntax

```
//Class def
public class <<class identifier>> extends <<other class>>
{
        …
        public <<class identifier>>()
        {
                super();//Call other class' default constructor
        }
}
```

Example

```
public class Employee extends Person
{
     protected int id;
     public Employee()
     {
          super();//Call to Person's def constr
          this.id = 0;
     }
     …
     public boolean equals(Employee e)
     {
          return e != null && super.equals(e) && this.id == e.getID();
     }
}
```

Design Diagram



Inheritance UML Class Diagram

- Arrows
  - Block arrow is the Inheritance or "is a" relationship
- "#" means "protected" scope
- Following down the *tree* means more specific
  - "Ugrad *is a* Student *is a* Person" ==
  - "Ugrad *is a* more specific Student who has a level, and a Student *is a* more specific person with a Student ID" ==
  - "Ugrad has the data level, ID (inherited from Student), name + favorite number (inherited from Person).

File Edit View Help

Related Example

File Edit View Help

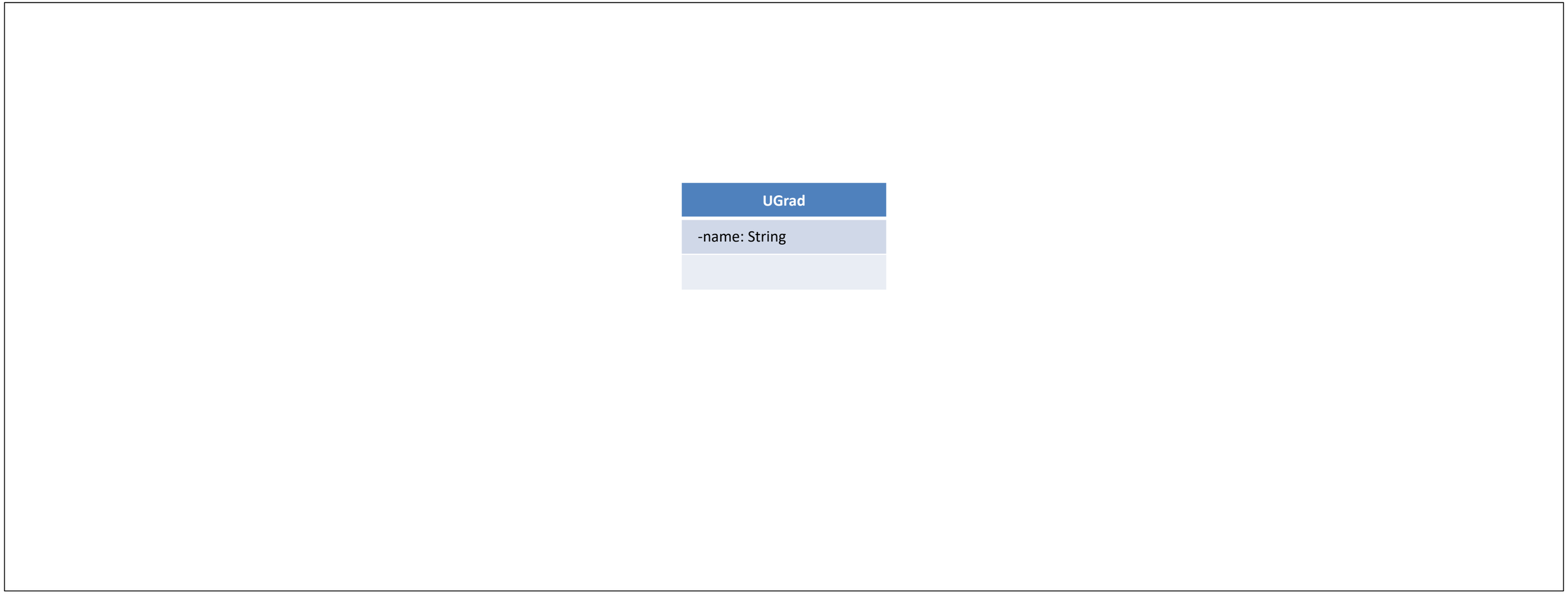
Personnel System

- 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

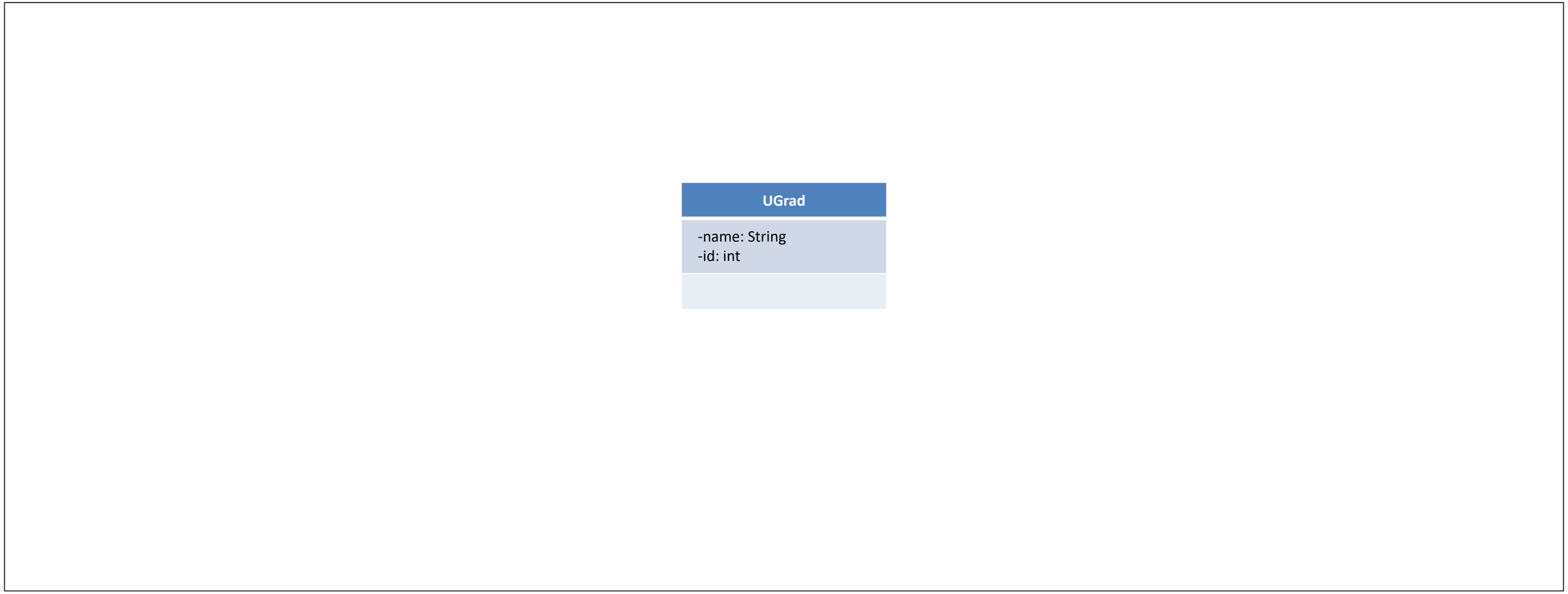
File Edit View Help

Design



File Edit View Help

Design



File Edit View Help

Design

File Edit View Help

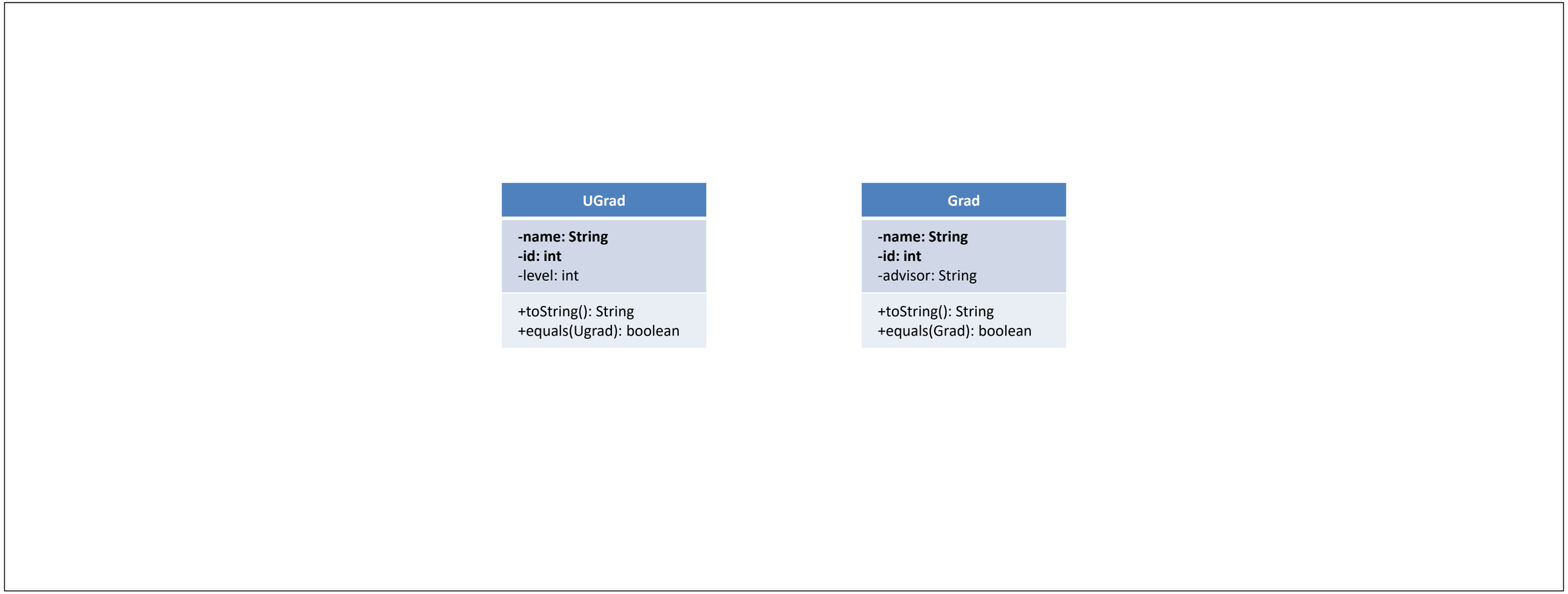
Design

File Edit View Help

Design

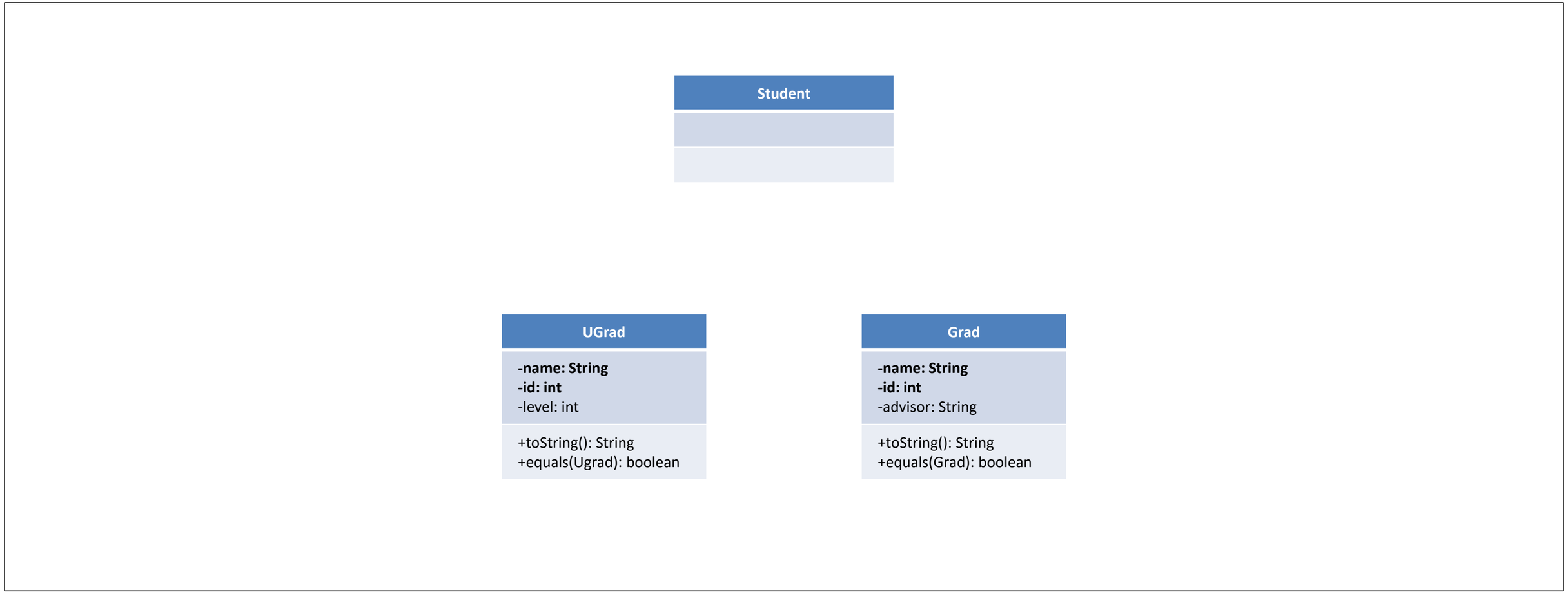
File Edit View Help

Design



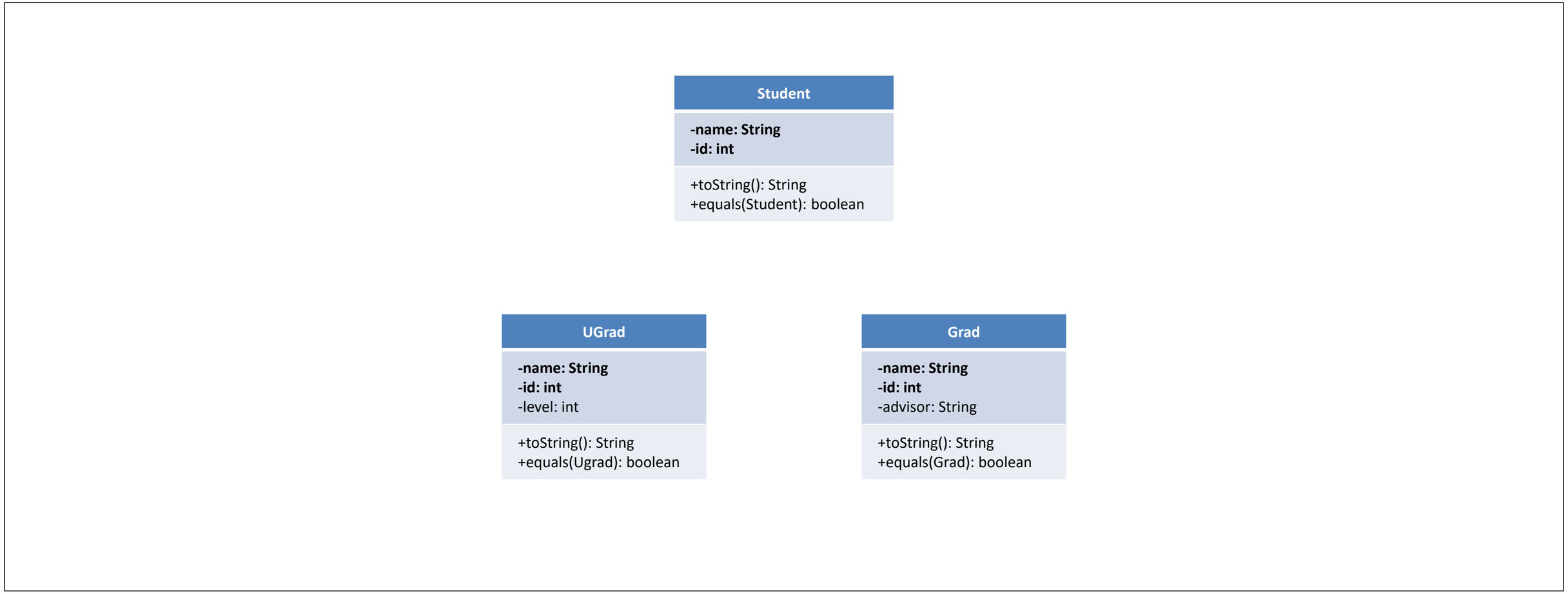
File Edit View Help

Design



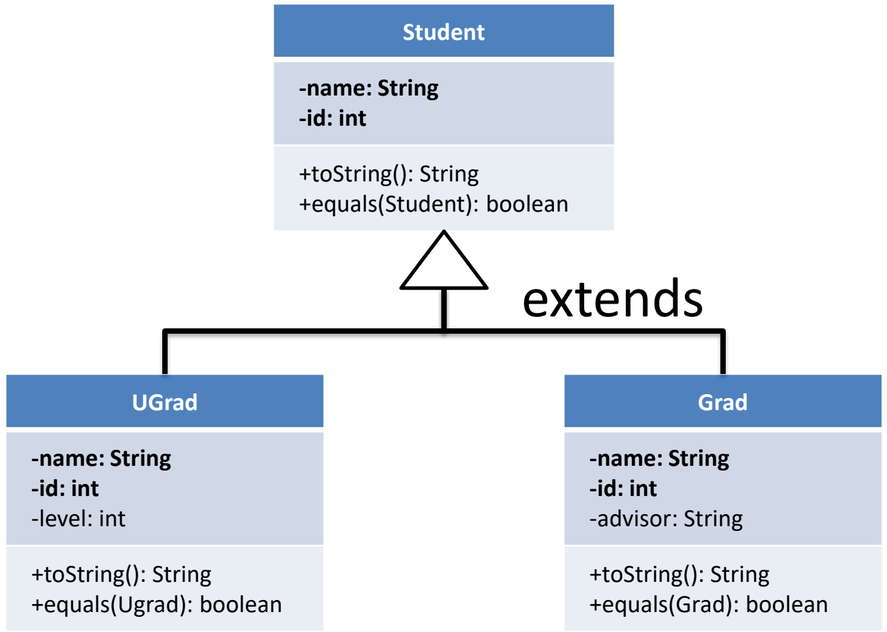
File Edit View Help

Design



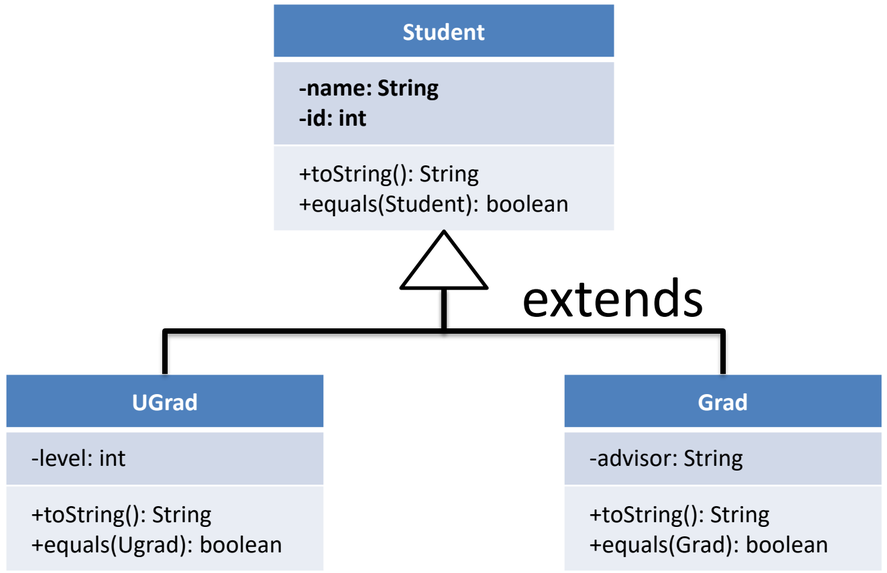
File Edit View Help

Design



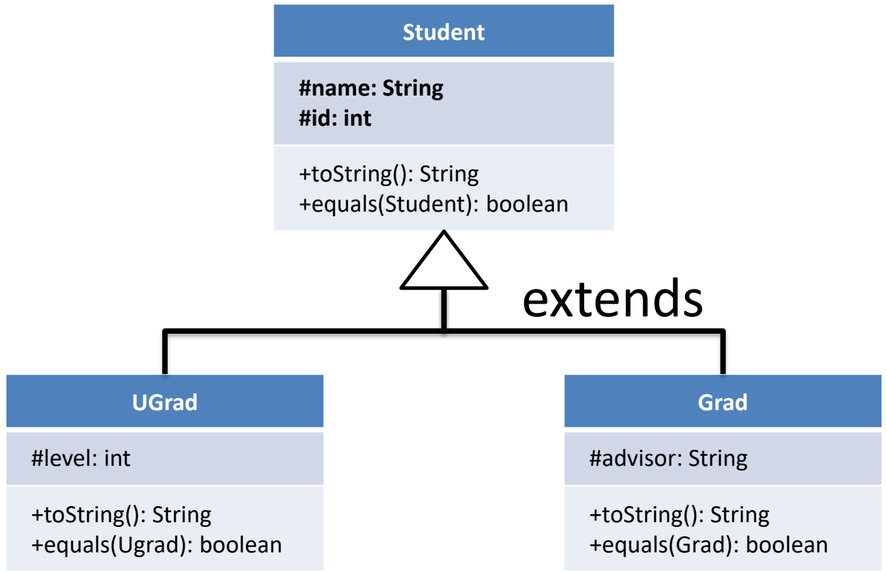
File Edit View Help

Design



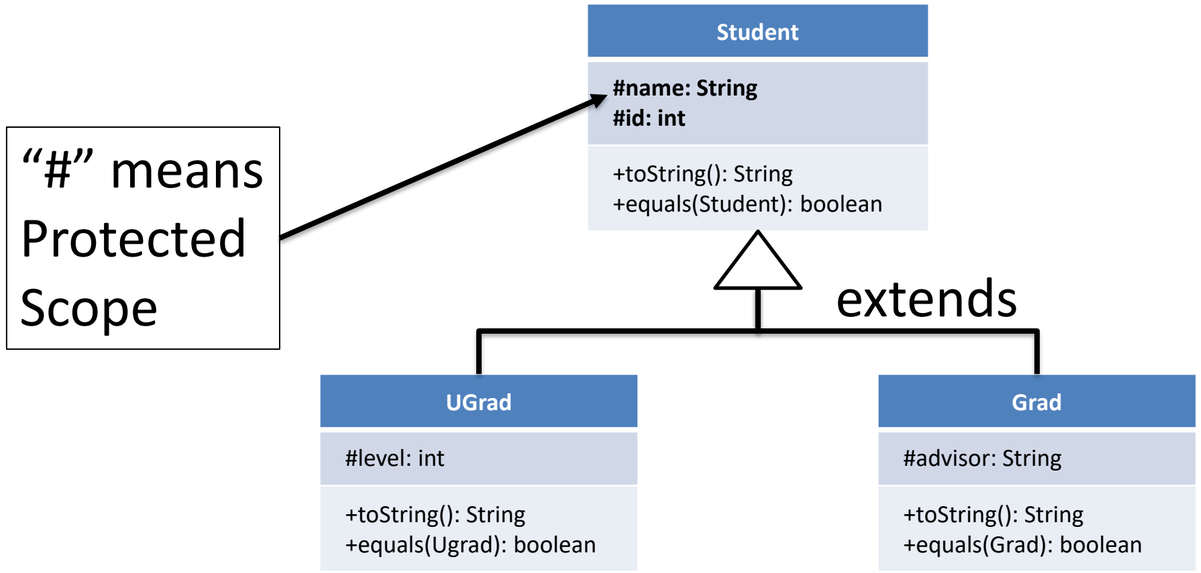
File Edit View Help

Design



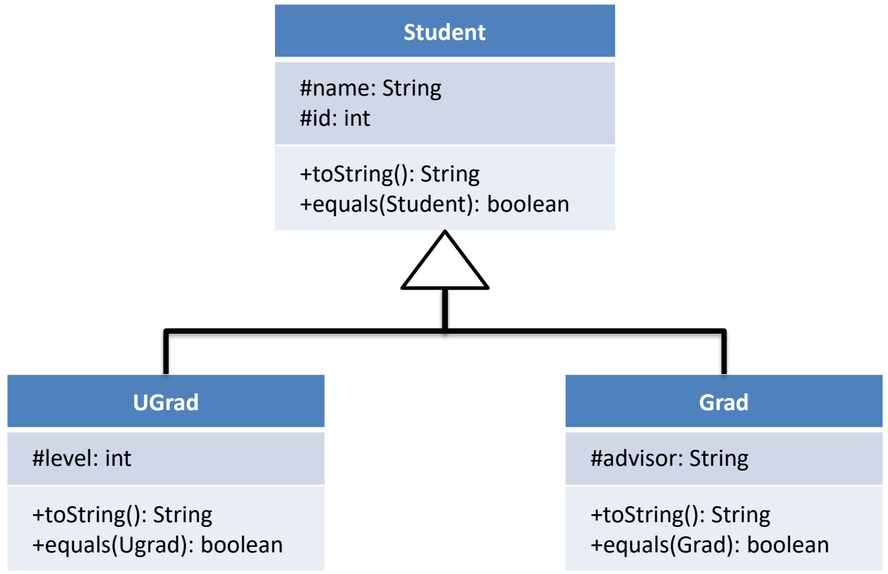
File Edit View Help

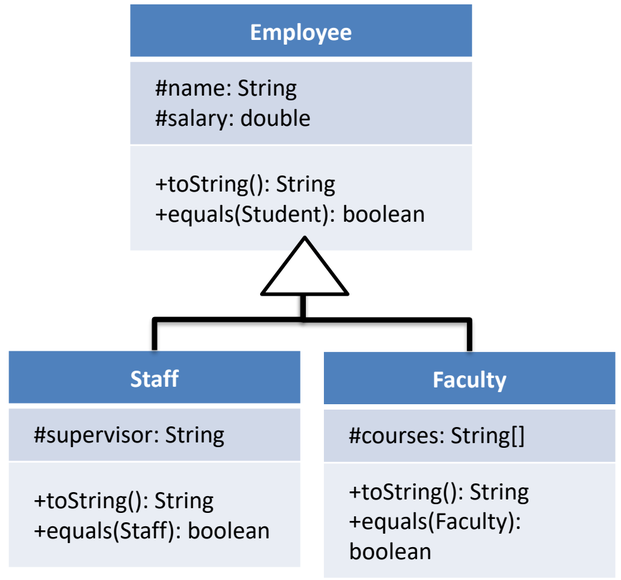
Design



File Edit View Help

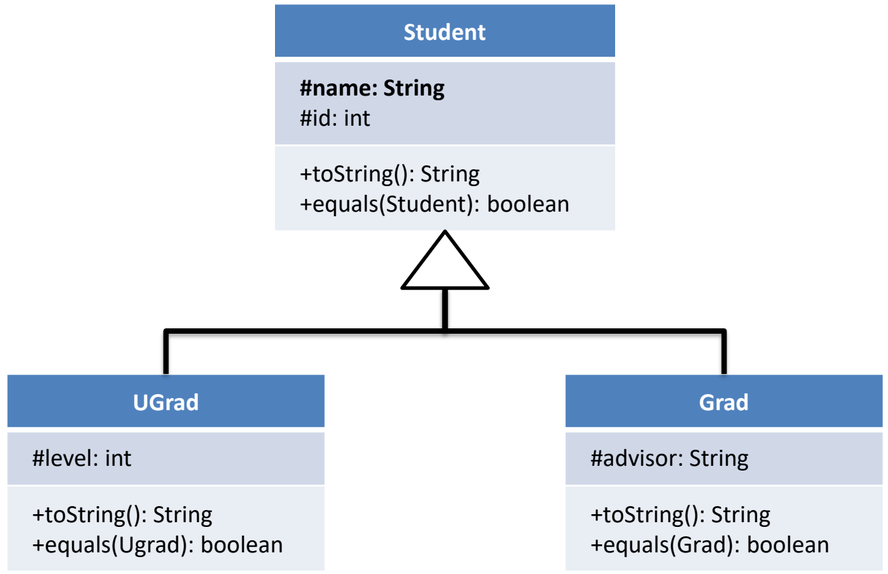
Design

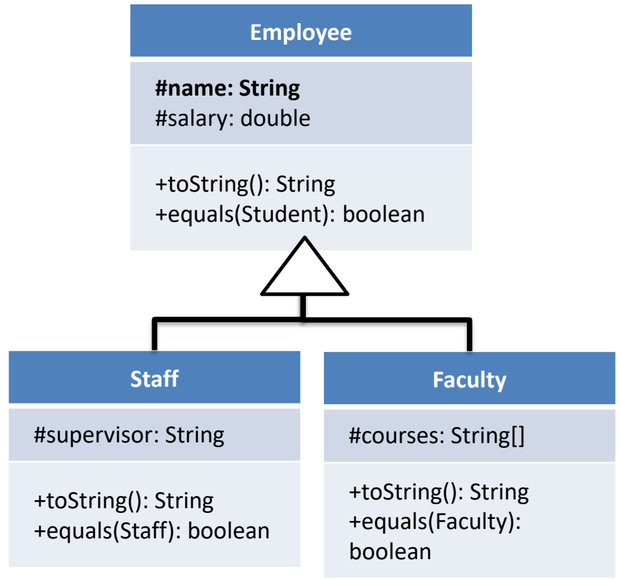




File Edit View Help

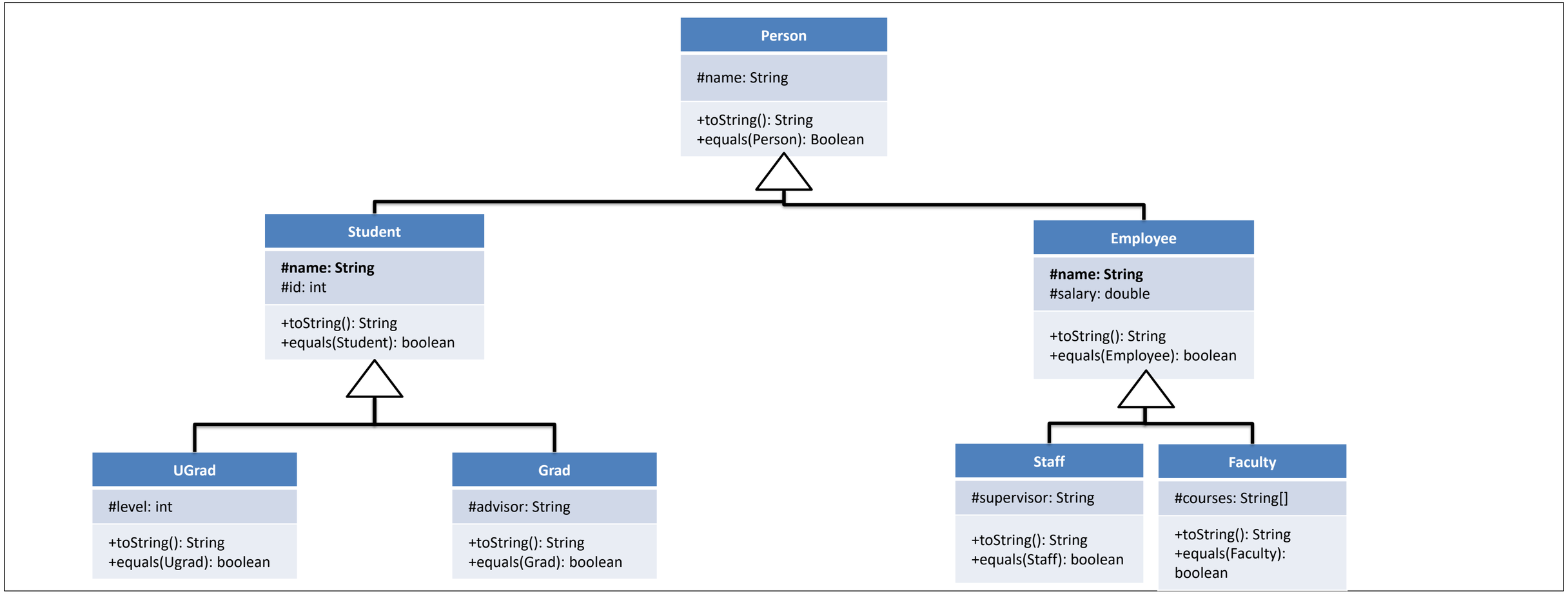
Design





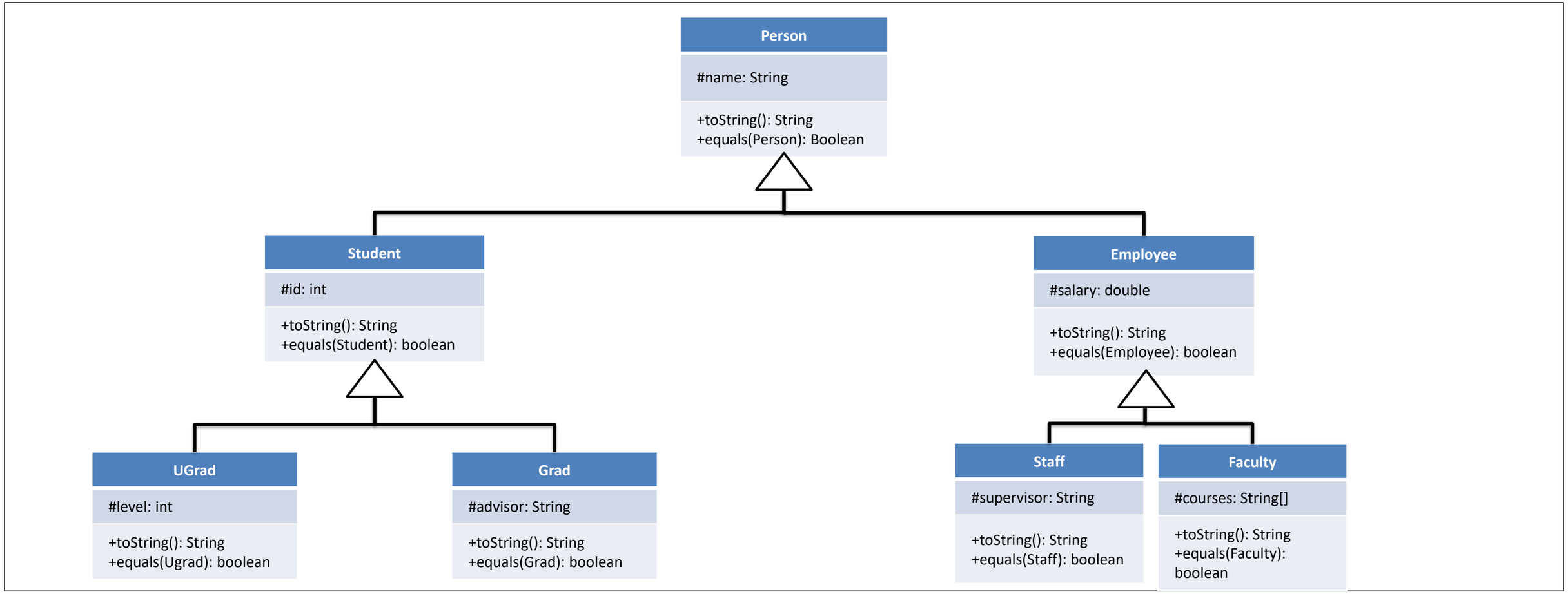
File Edit View Help

Design



File Edit View Help

Design



File Edit View Help

REQUIREMENTS UPDATE!!!

File Edit View Help

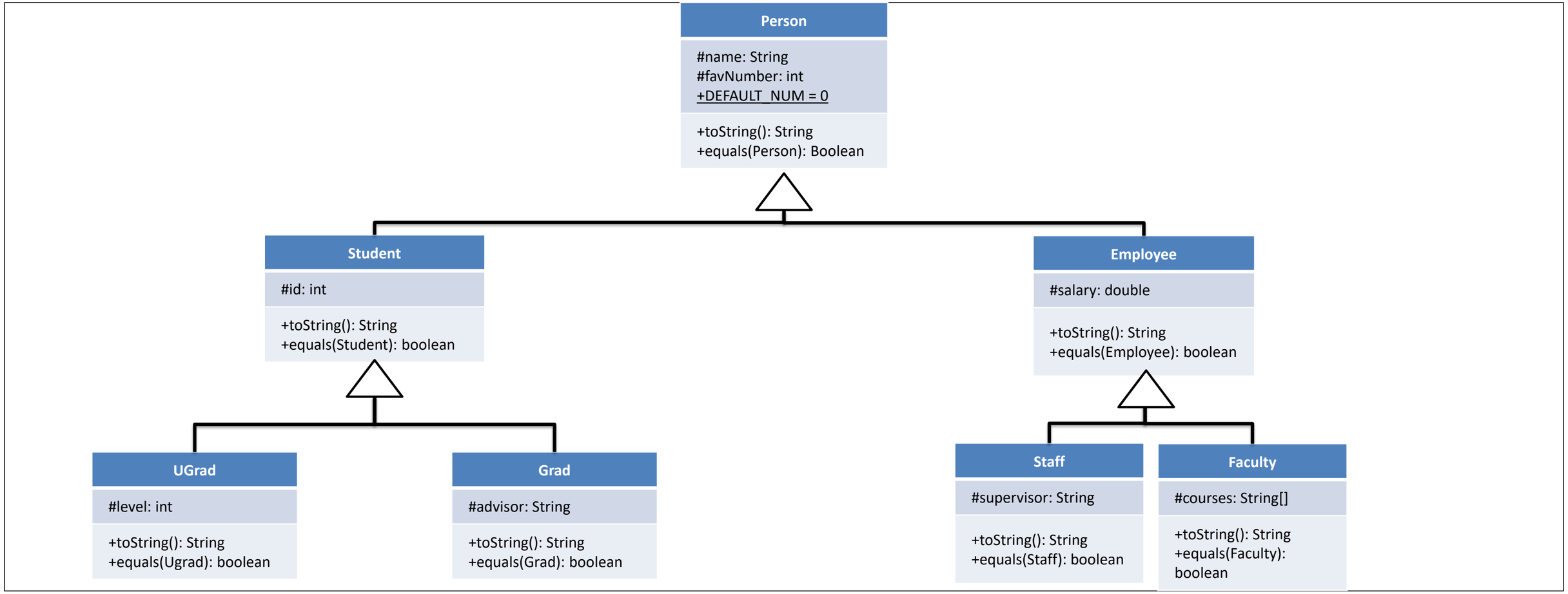
Personnel System

- 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
  - –Favorite Number
- Graduate Information
  - –Name
  - –Student ID
  - –Advisor's Name
  - –Favorite Number

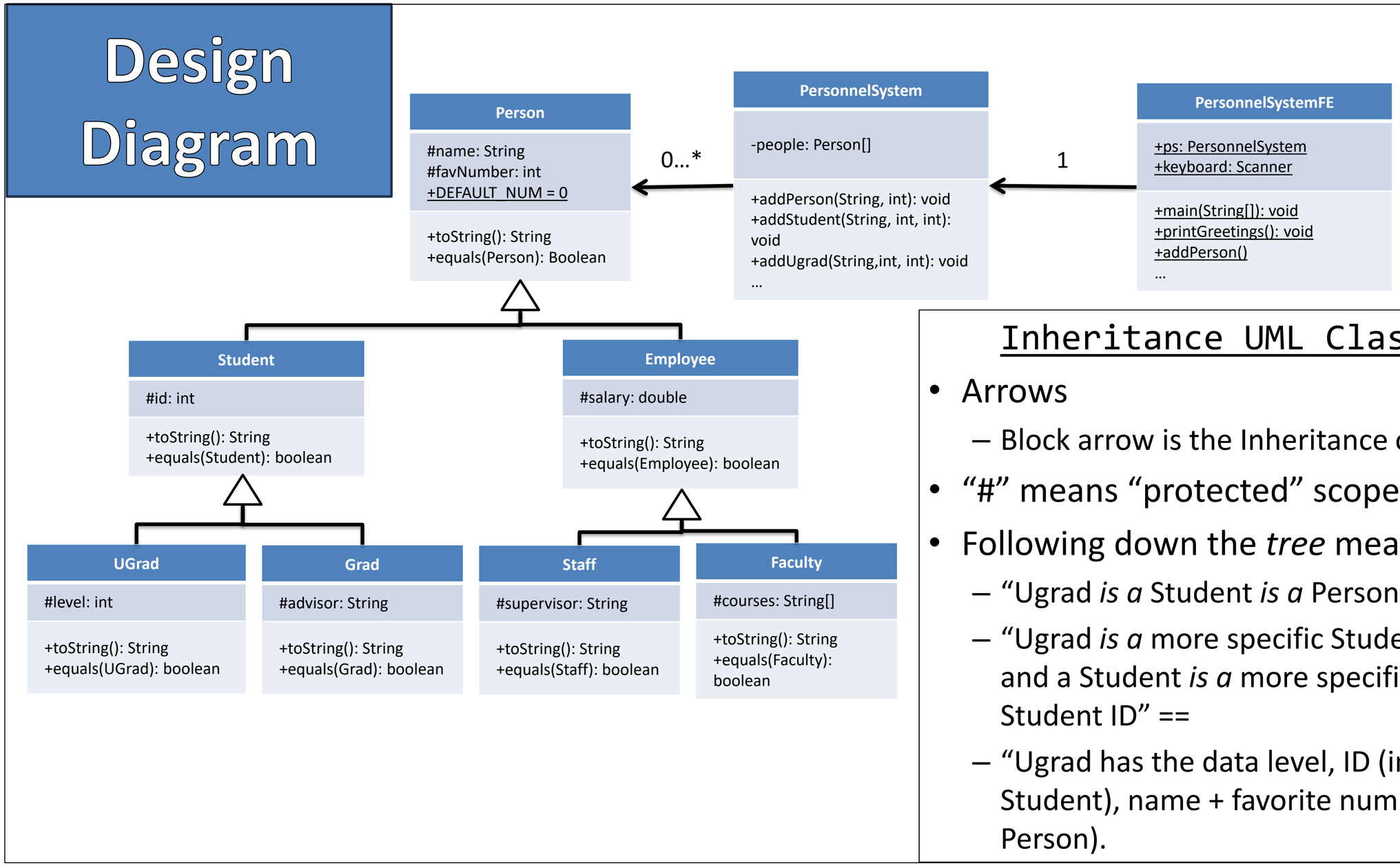
- Faculty Information
  - – Name
  - – Salary
  - – Courses
  - – Favorite Number
- Staff Information
  - – Name
  - – Salary
  - – Supervisor
  - – Favorite Number
- Should be able to:
  - – Add new people
  - – Remove people
  - – View all people in the system
- Clear and Easy-to-Use Frontend

File Edit View Help

Design



Design Diagram



Inheritance UML Class Diagram

- Arrows
  - Block arrow is the Inheritance or "is a" relationship
- "#" means "protected" scope
- Following down the *tree* means more specific
  - "Ugrad *is a* Student *is a* Person" ==
  - "Ugrad *is a* more specific Student who has a level, and a Student *is a* more specific person with a Student ID" ==
  - "Ugrad has the data level, ID (inherited from Student), name + favorite number (inherited from Person).

File Edit View Help

Example Implementation

Polymorphism

- "One becomes many" / "One action done many ways"
  - Similar to inheritance but focuses more on what an object does (its methods).
- "This *is a* that, which *DOES something*"
- One type's methods could be *implemented* in several ways
- **(Programming) Interfaces**
  - Similar in concept to **User Interfaces (UI)**, but are not the same.
  - Non-constructible datatype
  - Only method declarations
  - Provides "rules" for other programmers
  - Provides a *variable* for classes that *implement* the interface
  - Interface identifier must match the filename
- Classes "implement" interfaces
  - Unlike inheritance, a class may implement any number of interfaces
  - All methods in the interface MUST BE implemented or else there will be a syntax error
- Interfaces may "extend" other interfaces

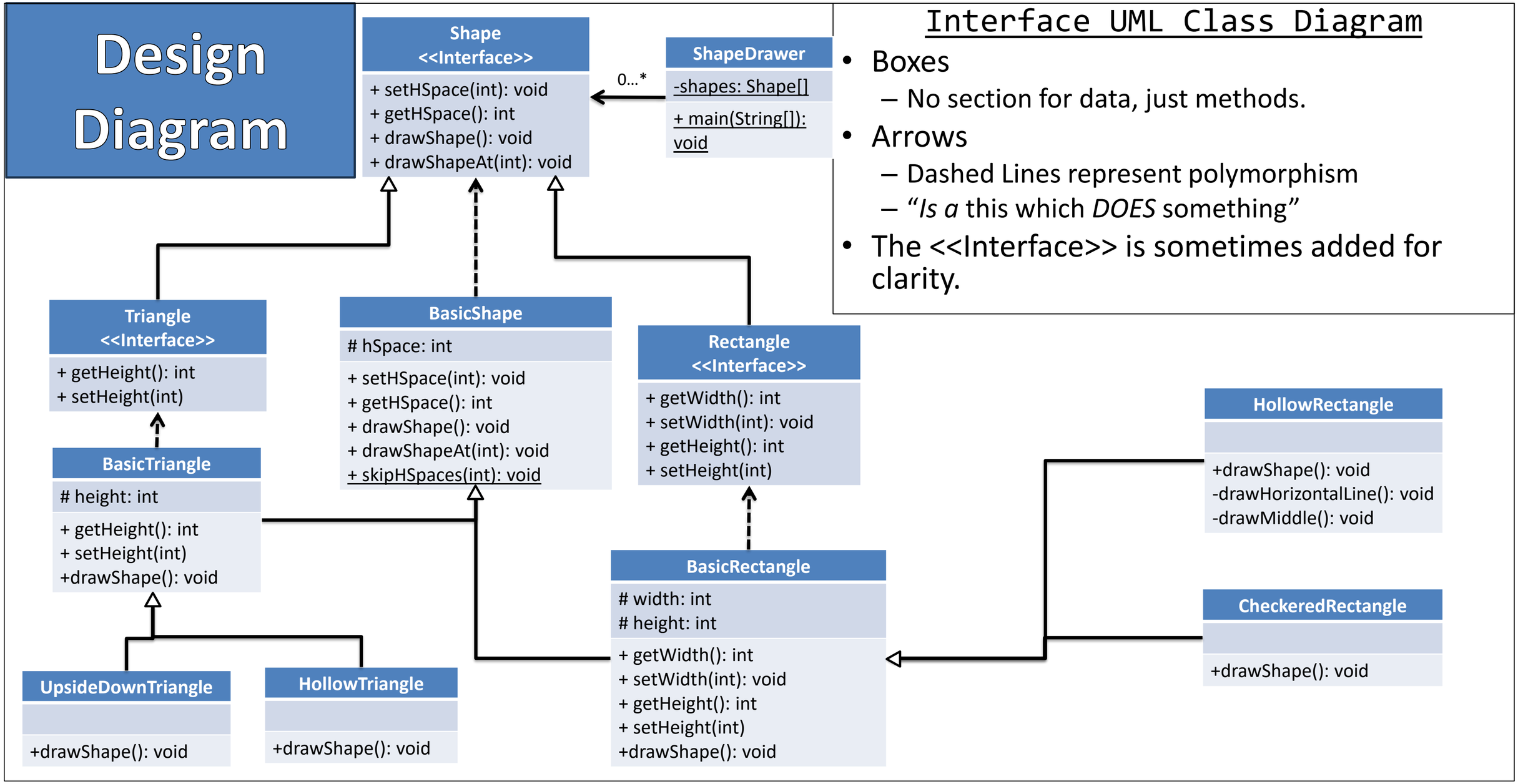
Syntax

```
//Interface def
public interface <<interface identifier>>
{
        <<method definition>>;//Semi-colon after each method
def
}
//Class implementing the interface
public class <<class identifier>> implements <<interface>>
{…
```

Example

```
public interface DrawableObject
{
      public void draw();
}
…
public class Rectangle implements DrawableObject
{
      …
      public void draw()
      {
            …
      }
}
```

Design Diagram



Interface UML Class Diagram

- Boxes
  - No section for data, just methods.
- Arrows
  - Dashed Lines represent polymorphism
  - "*Is a* this which *DOES* something"
- The <<Interface>> is sometimes added for clarity.

File Edit View Help

Related Example

File Edit View Help

Shape Drawing Program

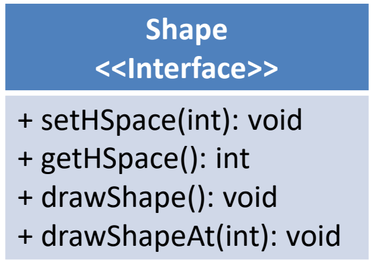
- Problem: We must create a program that can draw a variety of shapes in the console
- Draw Shapes in the console at set locations
  - Horizontal Spacing (Hs)
  - Vertical Spacing (Vs)
- Some Shapes mentioned were:
  - Rectangle
  - Triangle
  - Maybe more?

- Shapes could be drawn in a variety of ways
  - Filled
  - Hollow
  - Checkered Rectangle
  - Horizontal Striped Rectangle
  - Vertical Striped Rectangle
  - Upside Down Triangle (USD Tri)
  - Maybe More?

Example 2x3 Rectangle Hs=3 Vs=2	Example 3x3 Triangle Hs=4 Vs=1	Filled		Hollow		Checkered	H/V Striped		USD Tri	More?
| | Vs | ---*** ---*** Hs	| ----* ----** ----***	***** ***** ***** *****	* ** *** ****	***** * * * * *****	* ** * * ****	* * * * * * * * * *	***** ***** 	* * * * * * * * * * * *	**** *** ** *	???????? ???????? ???????? ????????

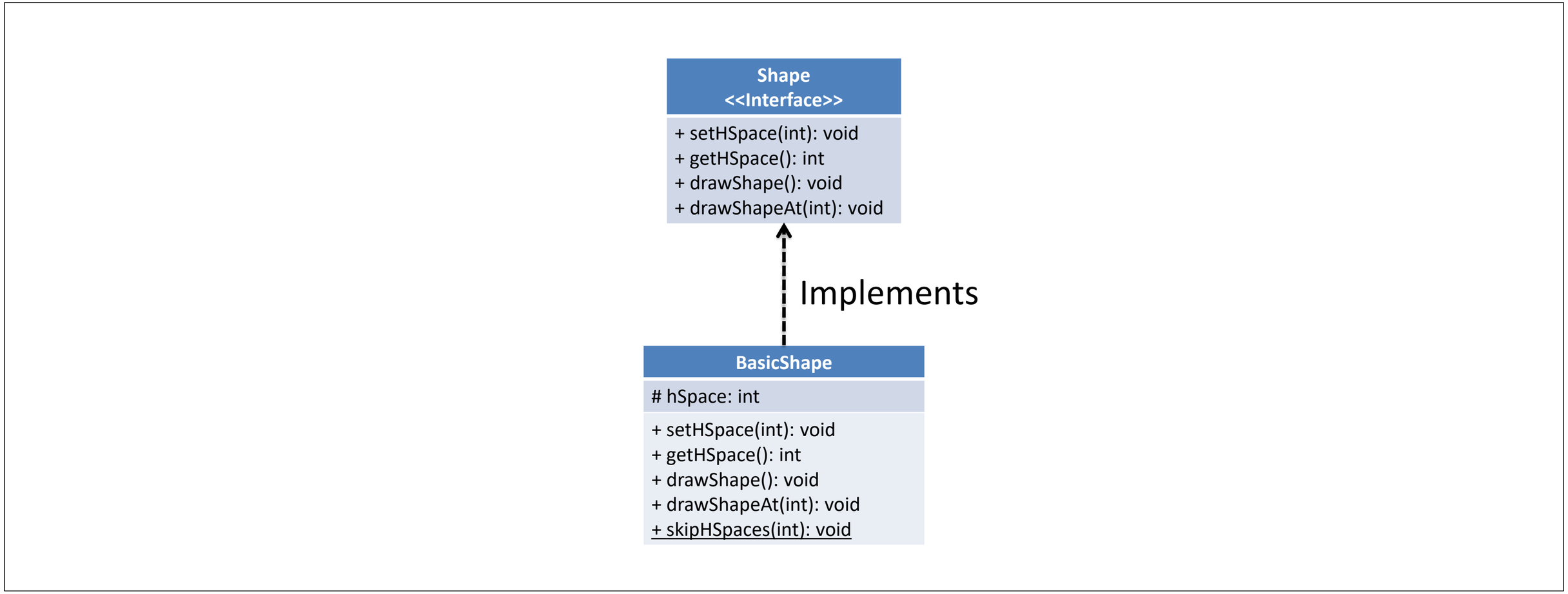
File Edit View Help

Shape Drawing Program



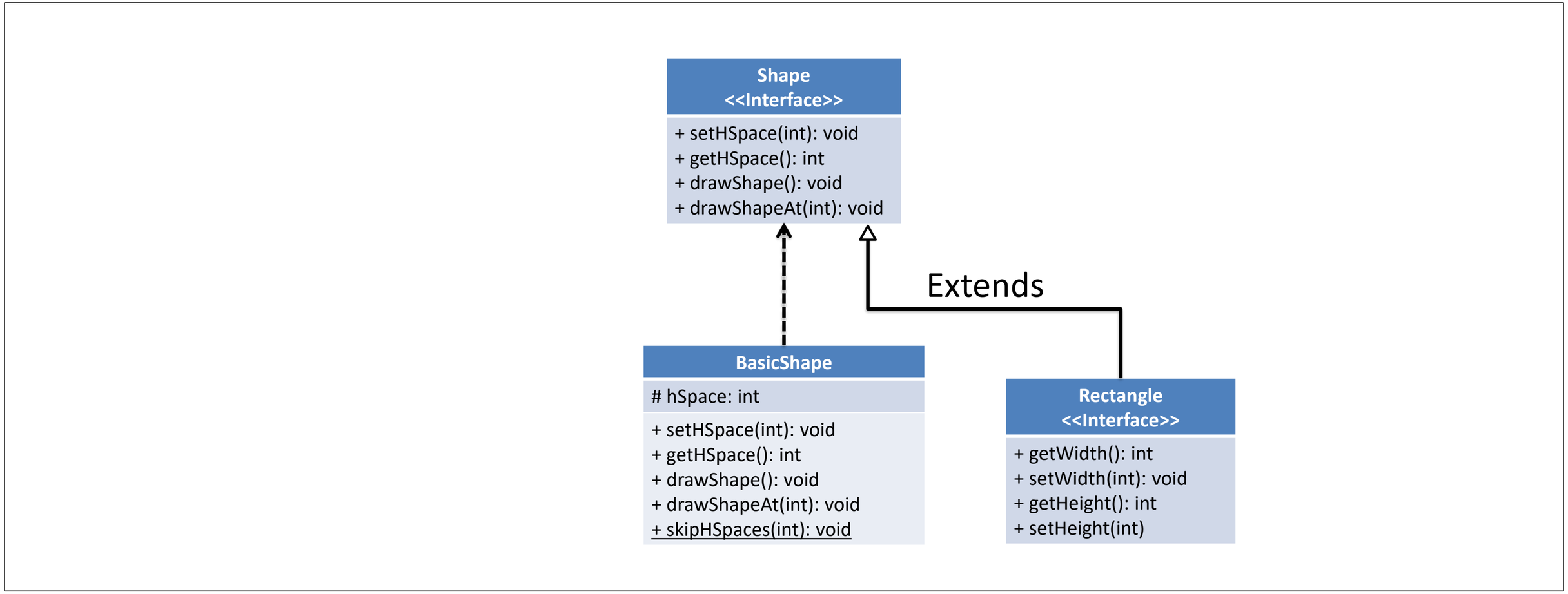
File Edit View Help

Shape Drawing Program



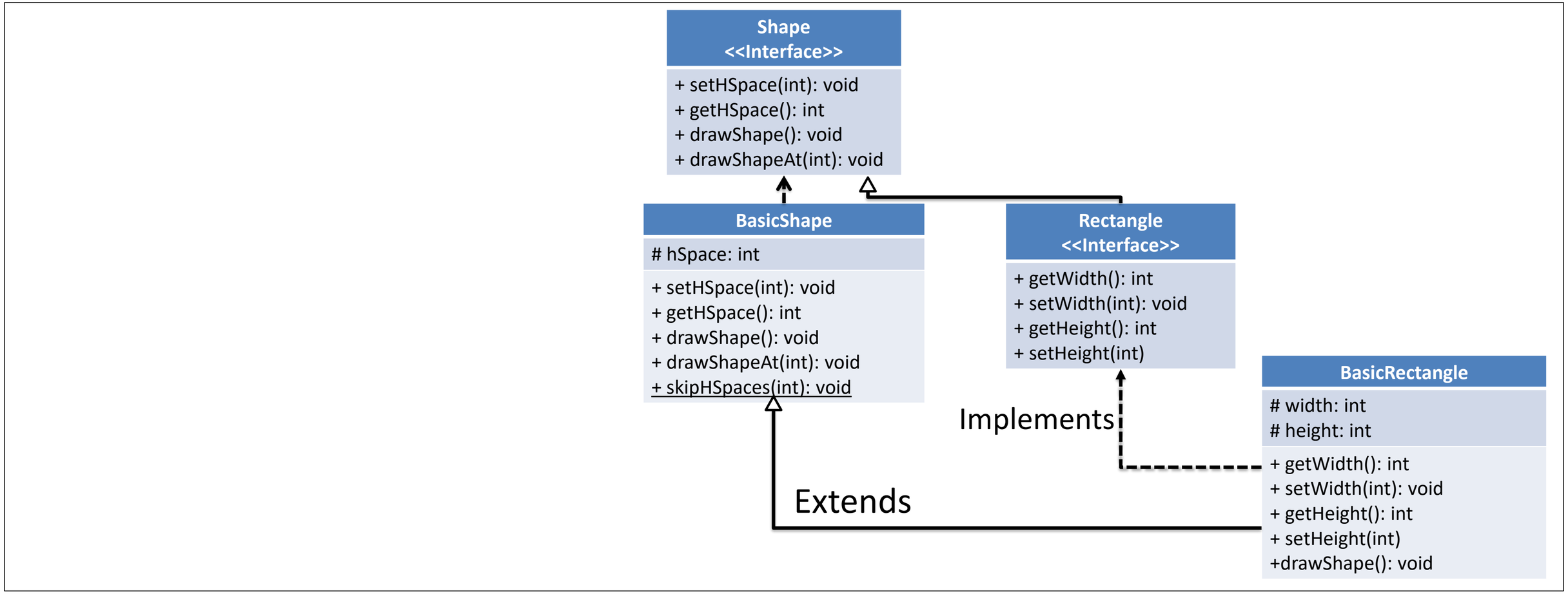
File Edit View Help

Shape Drawing Program

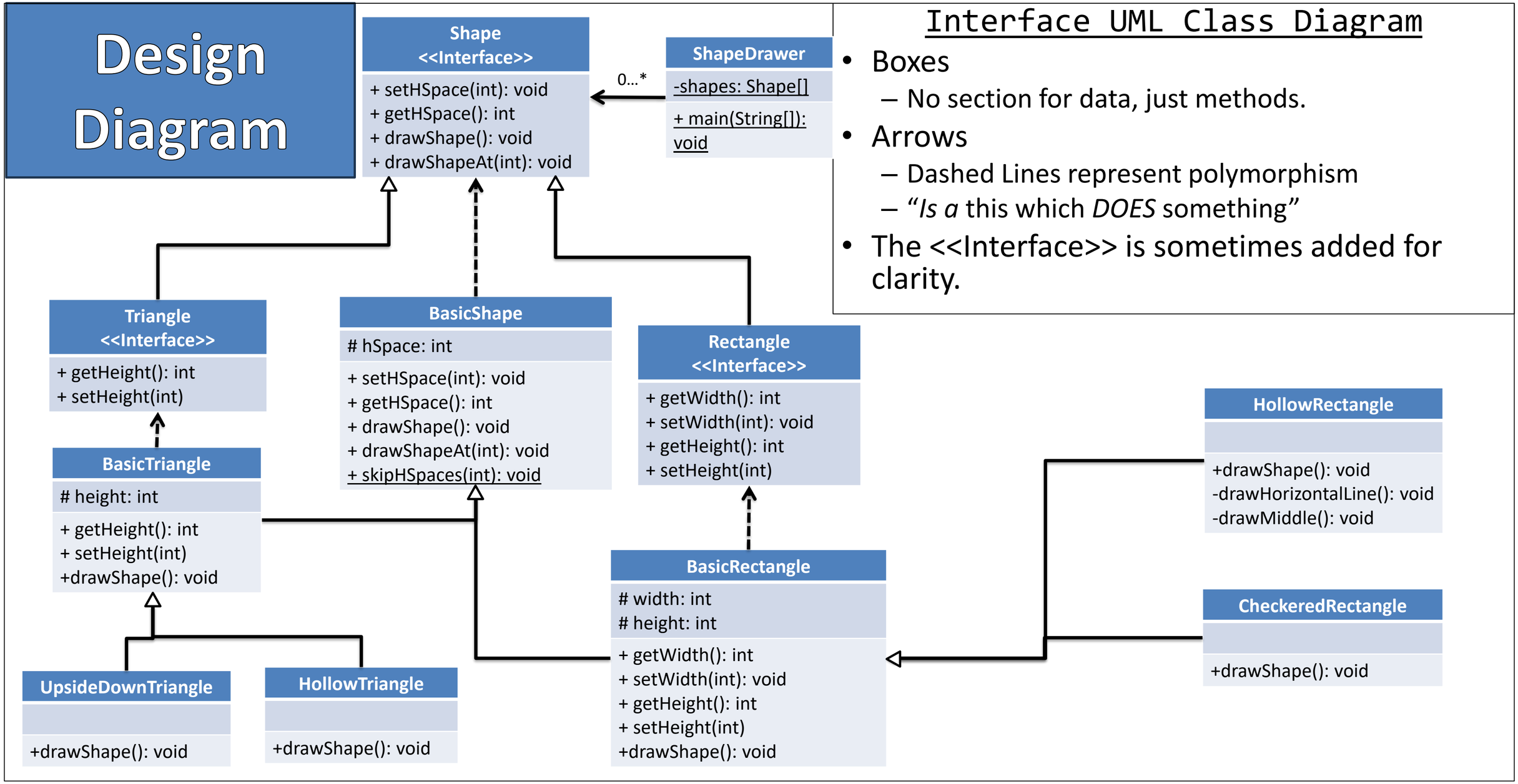


File Edit View Help

Shape Drawing Program



Design Diagram



Interface UML Class Diagram

- Boxes
  - No section for data, just methods.
- Arrows
  - Dashed Lines represent polymorphism
  - "*Is a* this which *DOES* something"
- The <<Interface>> is sometimes added for clarity.

File Edit View Help

Example Implementation

File Edit View Help

Programming Review

Part 04