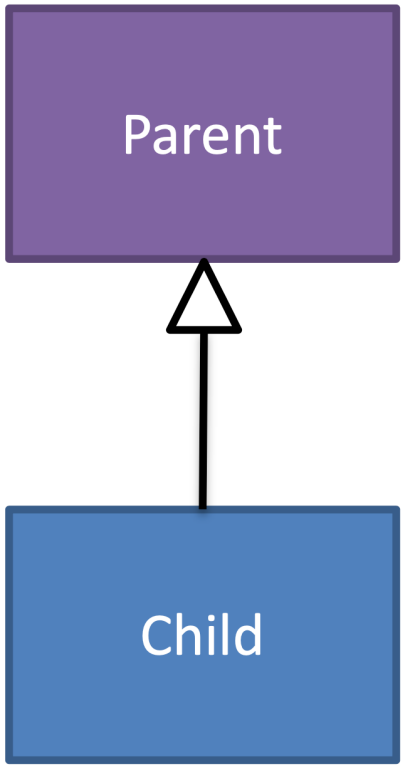
Inheritance and Polymorphism

Forest Agostinelli
University of South Carolina

Inheritance

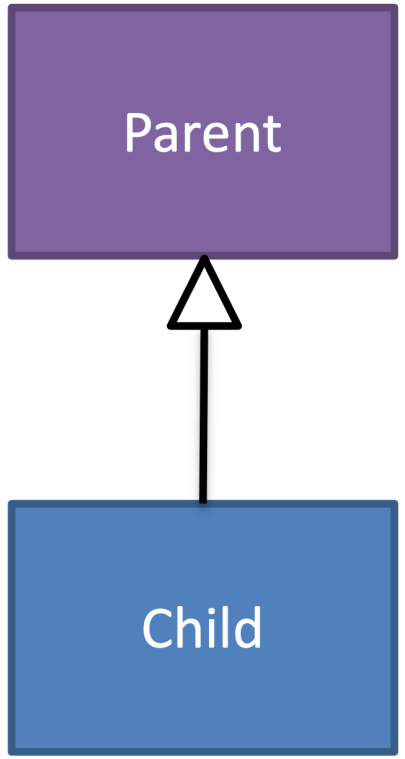
- Inheritance allows Data and Methods to be *inherited / absorbed* from one class into another
- In Java, this occurs between two classes
  - Subclass (Child): The class inheriting from another
  - Superclass (Parent): The class that is being inherited
- This is great for *extending* the properties and 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

- 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

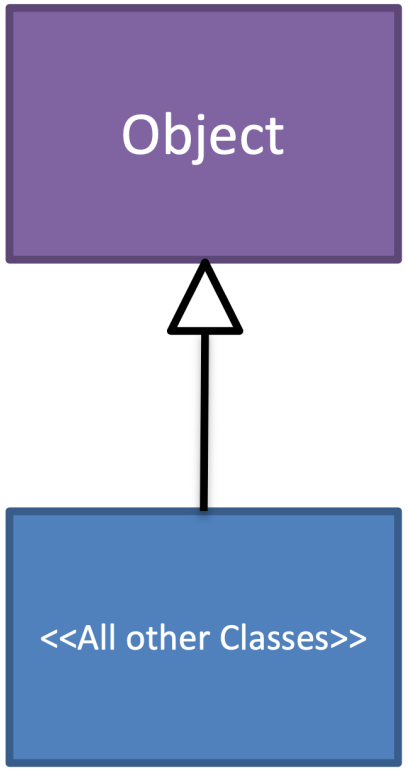
Inheritance Concept



Inheritance

- 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



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

- 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

- 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
```

- Here it overloads equals but does not override it (different signatures)

```java
/*
 * Written by JJ Shepherd
 */
public class Person {
    private String name;
    public Person()
    {
        this.name = "none";
    }
    public Person(String aN)
    {
        this.setName(aN);
    }
    public String getName()
    {
        return this.name;
    }
    public void setName(String aN)
    {
        if(aN != null)
            this.name = aN;
        else
            this.name = "none";
    }
    public String toString()
    {
        return "Name: "+this.name;
    }
    public boolean equals(Person aP)
    {
        return aP != null &&
                this.name.equals(aP.getName());
    }
}
```

```java
/*
 * Written by JJ Shepherd
 */
public class Student extends Person{
    private int id;
    public Student()
    {
        super();//Person's default constructor
        this.id = 0;
    }
    public Student(String aN, int anID)
    {
        super(aN);//Person's param constructor
        this.setID(anID);
    }
    public int getID()
    {
        return this.id;
    }
    public void setID(int anID)
    {
        if(anID >= 0)
            this.id = anID;
        else
            this.id = 0;
    }
    public String toString()
    {
        return super.toString()+" ID: "+this.id;
    }
    public boolean equals(Student aS)
    {
        return aS != null &&
                super.equals(aS) &&
                this.id == aS.getID();
    }
}
```

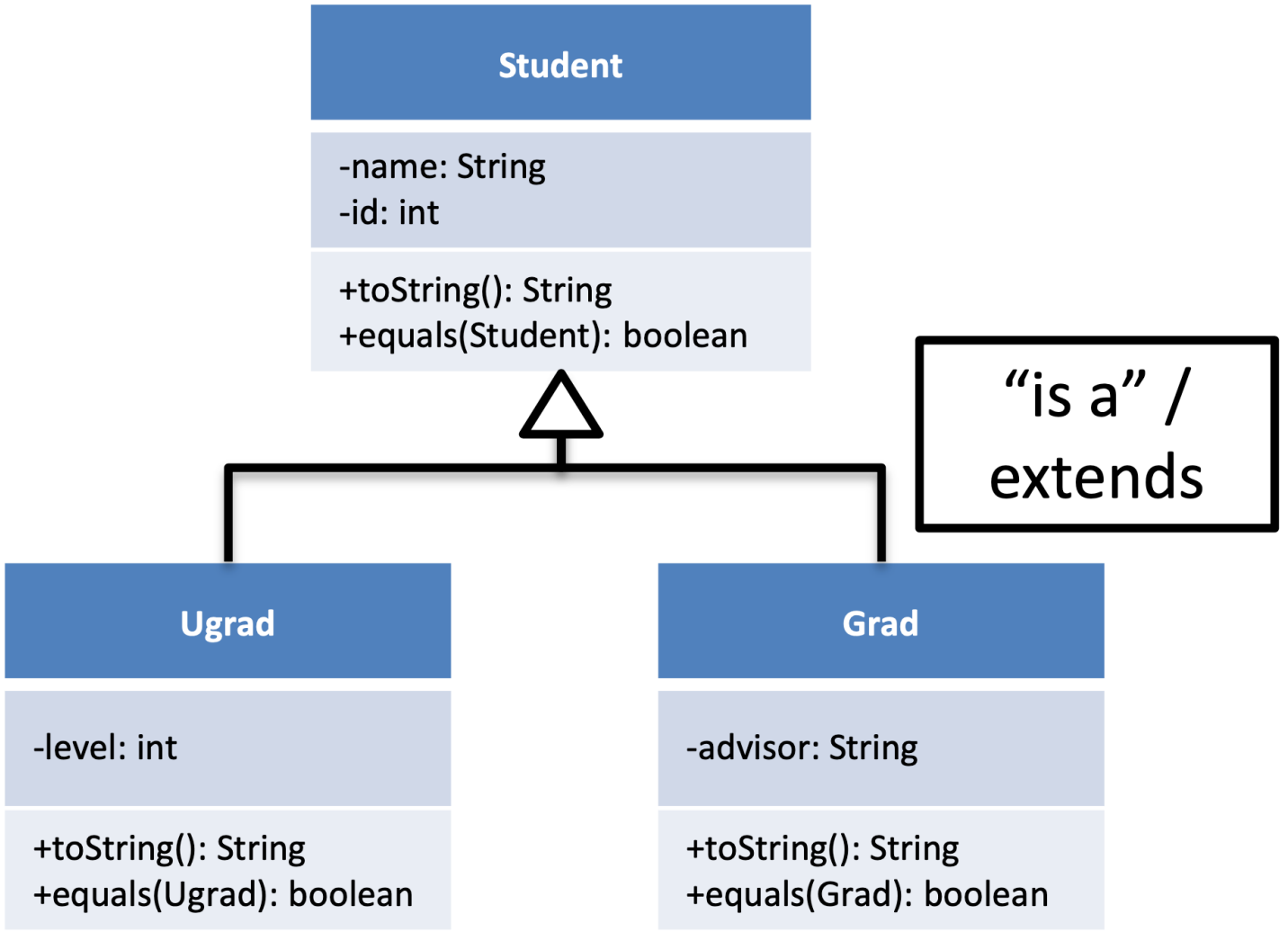
Quick Quiz: University System

- 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

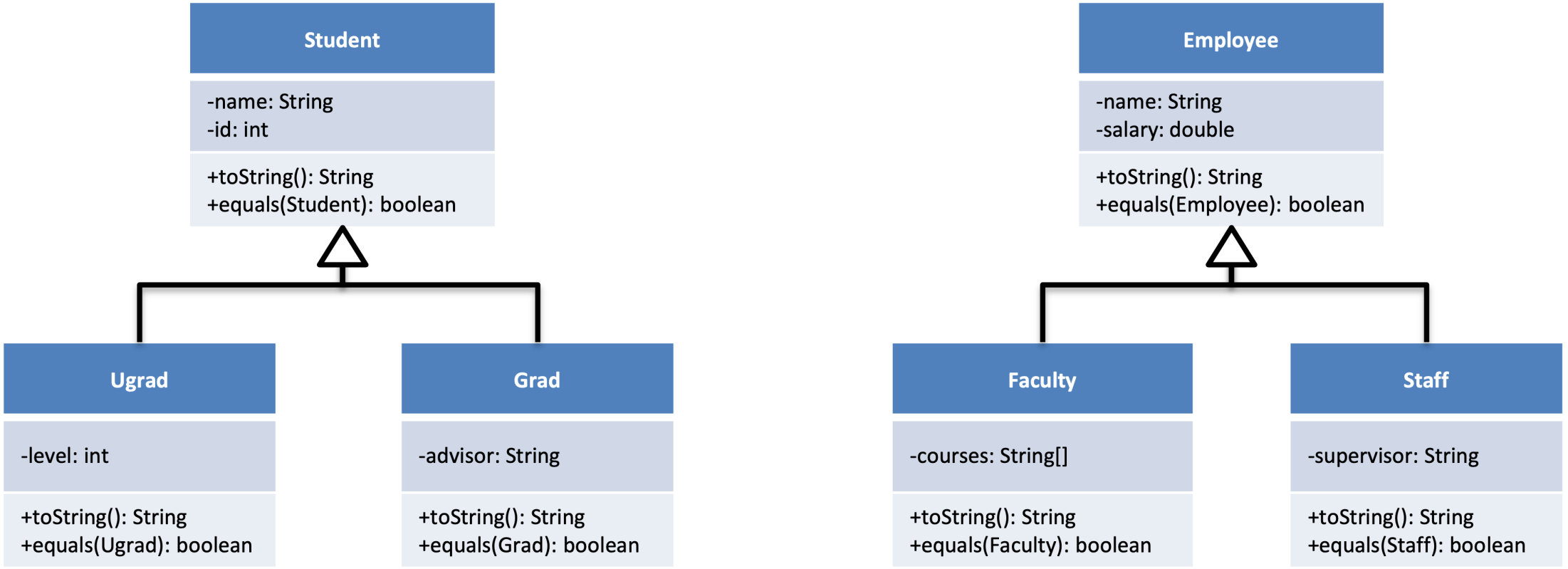
- Faculty Information
  - Name
  - Salary
  - Courses
- Staff Information
  - Name
  - Salary
  - Supervisor
- Should be able to:
  - Add new people
  - Remove people
  - View all people in the system
- Clear and Easy-to-Use Frontend

- How to design this?

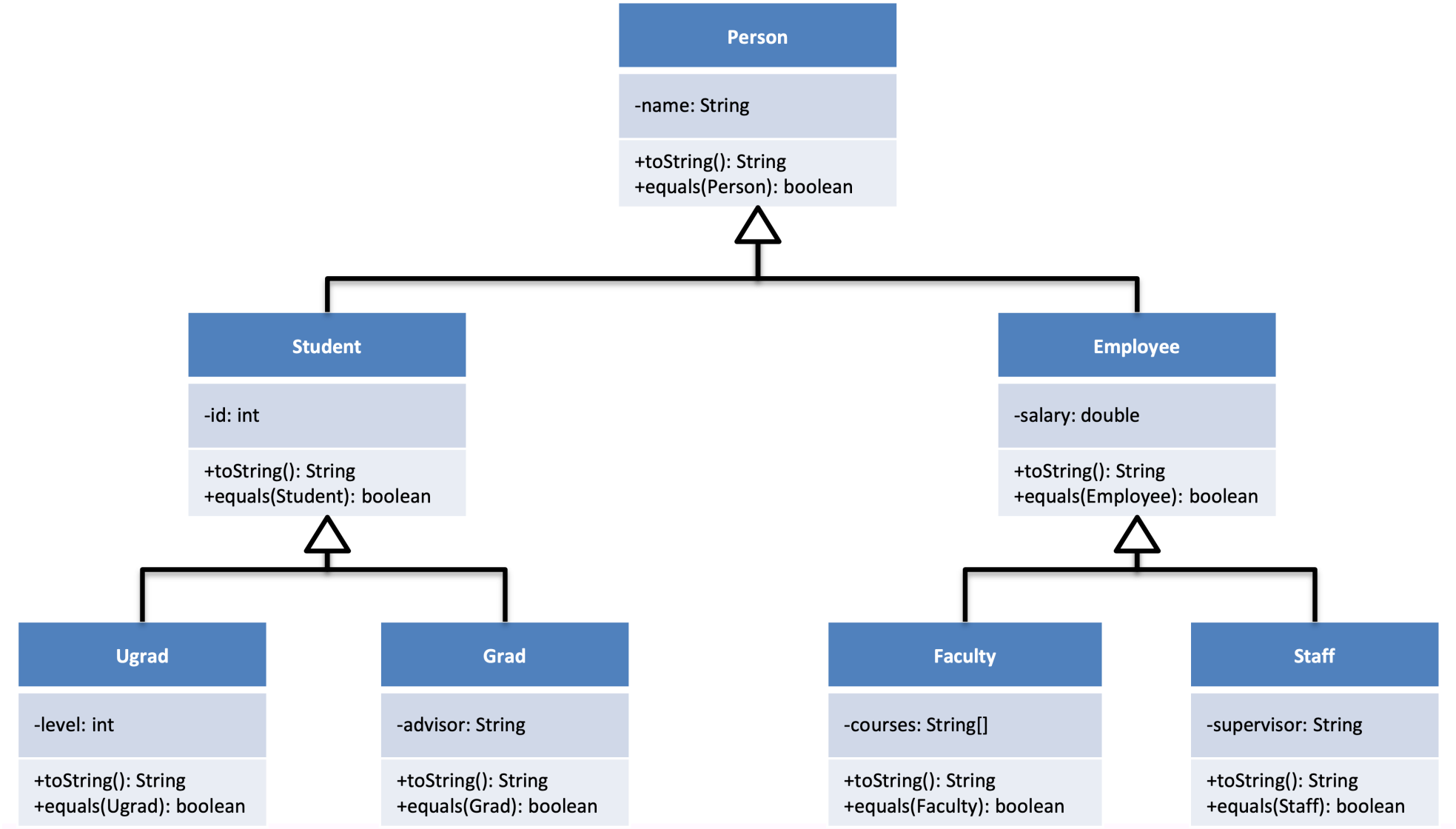
University System



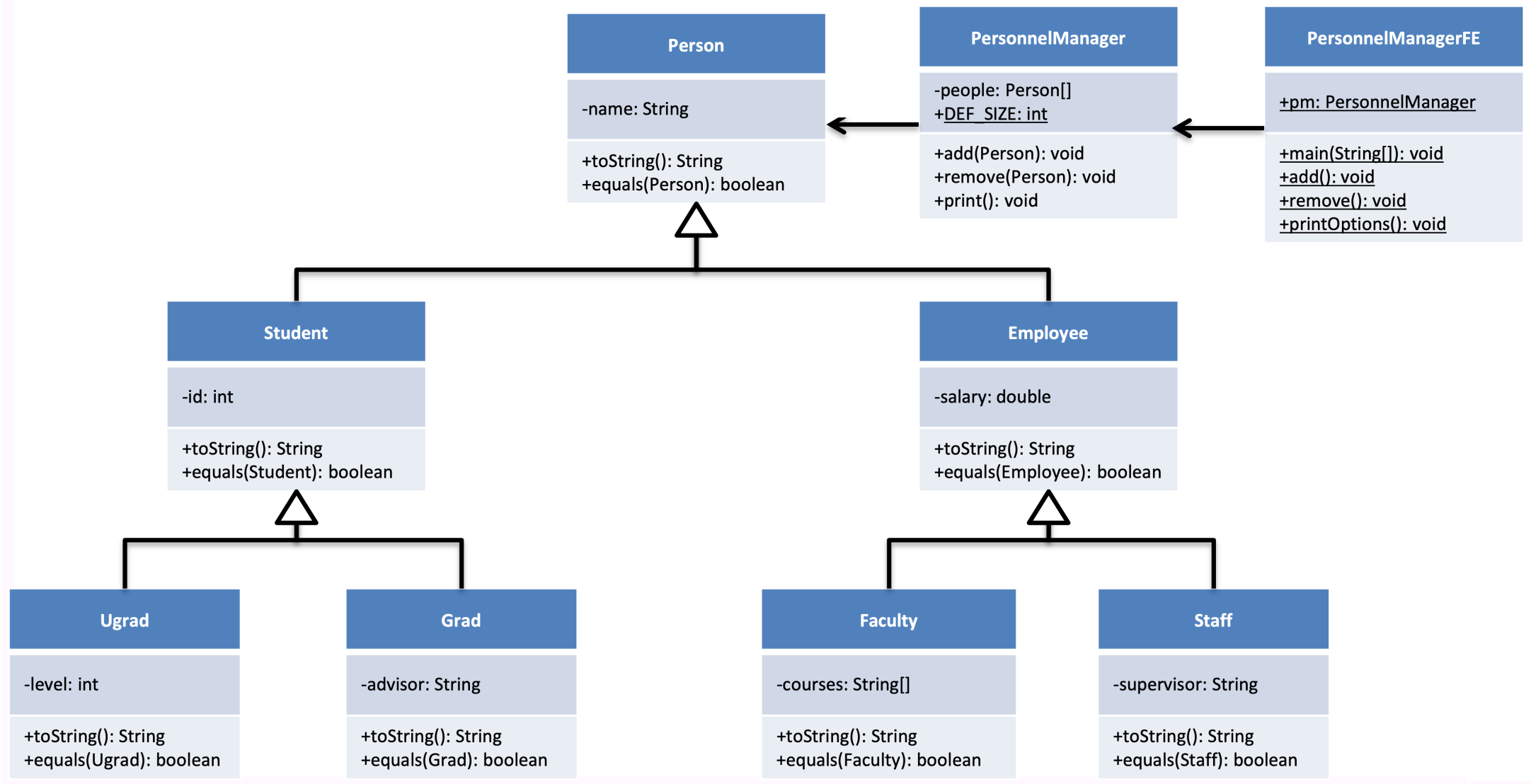
University System



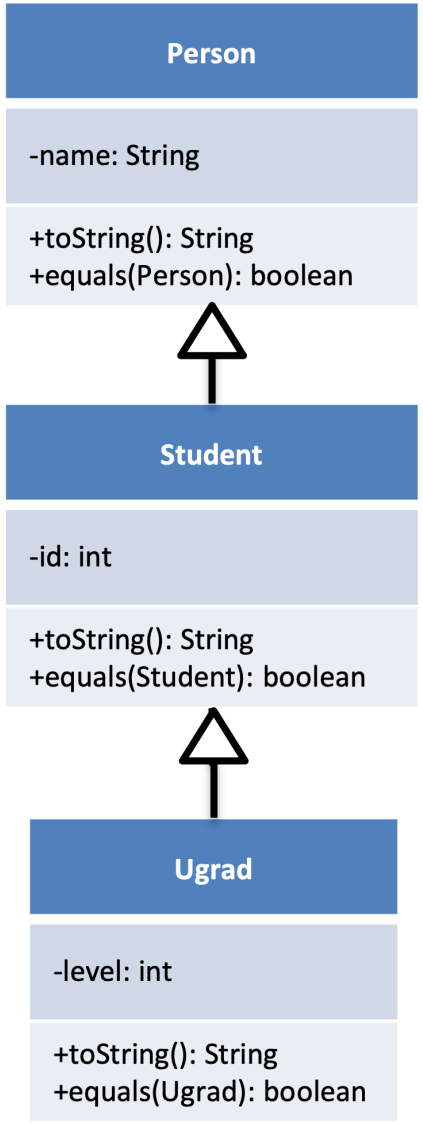
University System



University System



University System



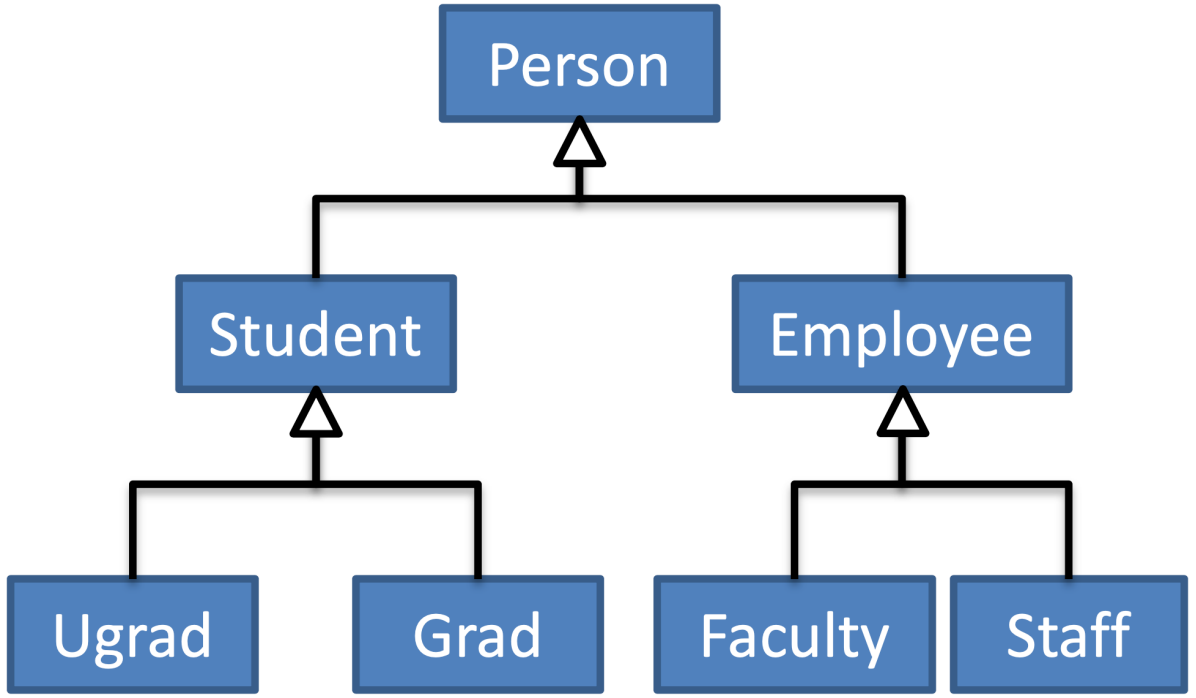
```java
/*
 * Written by JJ Shepherd
 */
public class Student extends Person{
    private int id;
    public Student()
    {
        super();//Person's default constructor
        this.id = 0;
    }
    public Student(String aN, int anID)
    {
        super(aN);//Person's param constructor
        this.setID(anID);
    }
    public int getID()
    {
        return this.id;
    }
    public void setID(int anID)
    {
        if(anID >= 0)
            this.id = anID;
        else
            this.id = 0;
    }
    public String toString()
    {
        return super.toString()+" ID: "+this.id;
    }
    public boolean equals(Student aS)
    {
        return aS != null &&
                super.equals(aS) &&
                this.id == aS.getID();
    }
}
```

```java
/*
 * Written by JJ Shepherd
 */
public class Ugrad extends Student{
    private int level;
    public Ugrad()
    {
        super();//Student's default constructor
        this.level = 1;
    }
    public Ugrad(String aN, int anID, int aL)
    {
        super(aN,anID);
        this.setLevel(aL);
    }
    public int getLevel()
    {
        return this.level;
    }
    public void setLevel(int aL)
    {
        if(aL>=1 && aL<=4)
            this.level = aL;
        else
            this.level = 1;
    }
    public String toString()
    {
        return super.toString()+" Level: "+this.level;
    }
    public boolean equals(Ugrad aU)
    {
        return aU != null &&
                super.equals(aU) &&
                this.level == aU.getLevel();
    }
}
```

University System

- "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



Quick Quiz

- One line is allowed, the other one is not. Why?

```
Person per = new Student("Bob", 0);
Student stu = new Person("Bob");
```

Compile Time and Run Time Polymorphism

- For overloaded methods, Java determines which method to call at compile time.
- Over overridden methods, Java determines which method to call at run time.

```
public class Person {
private String name;
…
public void orderTakeout(double numBurgers) {
    System.out.println(numBurgers + " BURGERS,
    55 FRIES, 55 TACOS, "
    + "55 PIES, 55 COKES, 100 TATER TOTS, 100
    PIZZAS, 100 TENDERS, "
    + "100 MEATBALLS, 100 COFFEES, 55 WINGS, 55
    SHAKES, 55 PANCAKES, "
    + "55 PASTAS, 55 PEPPERS AND 155 TATERS!");
    }
}
```

```
public class Student extends Person {
private int id;
…
public void orderTakeout(int numBurgers) {
        System.out.println(numBurgers + "
burgers please.");
}
}
```

```
Person per1 = new Person("Bob");
Student stu1 = new Student("Bob", 0);
Person[] people = new Person[2];
people[0] = per1;
people[1] = stu1;
per1.orderTakeout(55);
stu1.orderTakeout(55);
people[0].orderTakeout(55);
people[1].orderTakeout(55);
```

```java
public class Person {
private String name;
…
public void orderTakeout(double numBurgers) {
    System.out.println(numBurgers + " BURGERS,
    55 FRIES, 55 TACOS, "
    + "55 PIES, 55 COKES, 100 TATER TOTS, 100
    PIZZAS, 100 TENDERS, "
    + "100 MEATBALLS, 100 COFFEES, 55 WINGS, 55
    SHAKES, 55 PANCAKES, "
    + "55 PASTAS, 55 PEPPERS AND 155 TATERS!");
    }
}
```

```java
public class Student extends Person {
private int id;
…
public void orderTakeout(int numBurgers) {
        System.out.println(numBurgers + "
burgers please.");
}
}
```

```java
Person per1 = new Person("Bob");
Student stu1 = new Student("Bob", 0);
Person[] people = new Person[2];
people[0] = per1;
people[1] = stu1;
per1.orderTakeout(55);
stu1.orderTakeout(55);
people[0].orderTakeout(55);
people[1].orderTakeout(55);
```

```
55.0 BURGERS, 55 FRIES, 55 TACOS, 55 PIES, 55 COKES,
100 TATER TOTS, 100 PIZZAS, 100 TENDERS, 100 MEATBALLS,
100 COFFEES, 55 WINGS, 55 SHAKES, 55 PANCAKES, 55
PASTAS, 55 PEPPERS AND 155 TATERS!

55 burgers please.

55.0 BURGERS, 55 FRIES, 55 TACOS, 55 PIES, 55 COKES,
100 TATER TOTS, 100 PIZZAS, 100 TENDERS, 100 MEATBALLS,
100 COFFEES, 55 WINGS, 55 SHAKES, 55 PANCAKES, 55
PASTAS, 55 PEPPERS AND 155 TATERS!

55.0 BURGERS, 55 FRIES, 55 TACOS, 55 PIES, 55 COKES,
100 TATER TOTS, 100 PIZZAS, 100 TENDERS, 100 MEATBALLS,
100 COFFEES, 55 WINGS, 55 SHAKES, 55 PANCAKES, 55
PASTAS, 55 PEPPERS AND 155 TATERS!
```

```
public class Person {
private String name;
…
public void orderTakeout(double numBurgers) {
    System.out.println(numBurgers + " BURGERS,
    55 FRIES, 55 TACOS, "
    + "55 PIES, 55 COKES, 100 TATER TOTS, 100
    PIZZAS, 100 TENDERS, "
    + "100 MEATBALLS, 100 COFFEES, 55 WINGS, 55
    SHAKES, 55 PANCAKES, "
    + "55 PASTAS, 55 PEPPERS AND 155 TATERS!");
    }
}
```

```
public class Student extends Person {
private int id;
…
public void orderTakeout(double numBurgers) {
        System.out.println(numBurgers + "
burgers please.");
}
}
```

```
Person per1 = new Person("Bob");
Student stu1 = new Student("Bob", 0);
Person[] people = new Person[2];
people[0] = per1;
people[1] = stu1;
per1.orderTakeout(55);
stu1.orderTakeout(55);
people[0].orderTakeout(55);
people[1].orderTakeout(55);
```

```java
public class Person {
private String name;
…
public void orderTakeout(double numBurgers) {
    System.out.println(numBurgers + " BURGERS,
    55 FRIES, 55 TACOS, "
    + "55 PIES, 55 COKES, 100 TATER TOTS, 100
    PIZZAS, 100 TENDERS, "
    + "100 MEATBALLS, 100 COFFEES, 55 WINGS, 55
    SHAKES, 55 PANCAKES, "
    + "55 PASTAS, 55 PEPPERS AND 155 TATERS!");
    }
}
```

```java
public class Student extends Person {
private int id;
…
public void orderTakeout(double numBurgers) {
        System.out.println(numBurgers + "
burgers please.");
}
}
```

```java
Person per1 = new Person("Bob");
Student stu1 = new Student("Bob", 0);
Person[] people = new Person[2];
people[0] = per1;
people[1] = stu1;
per1.orderTakeout(55);
stu1.orderTakeout(55);
people[0].orderTakeout(55);
people[1].orderTakeout(55);
```

```
55.0 BURGERS, 55 FRIES, 55 TACOS, 55 PIES, 55 COKES,
100 TATER TOTS, 100 PIZZAS, 100 TENDERS, 100 MEATBALLS,
100 COFFEES, 55 WINGS, 55 SHAKES, 55 PANCAKES, 55
PASTAS, 55 PEPPERS AND 155 TATERS!

55 burgers please.

55.0 BURGERS, 55 FRIES, 55 TACOS, 55 PIES, 55 COKES,
100 TATER TOTS, 100 PIZZAS, 100 TENDERS, 100 MEATBALLS,
100 COFFEES, 55 WINGS, 55 SHAKES, 55 PANCAKES, 55
PASTAS, 55 PEPPERS AND 155 TATERS!

55 burgers please.
```

```java
/*
 * Written by JJ Shepherd
 */
public class Tester {

    public static void main(String[] args) {
        Person p = new Person();
        Student s = new Student();
        Ugrad u = new Ugrad();

        System.out.println(p);
        System.out.println(s);
        System.out.println(u);

        Person[] people = new Person[3];
        people[0] = new Person("asdf");
        people[1] = new Student("asdf2",4);
        people[2] = new Ugrad("asdf3",5,2);

        for(int i=0;i<people.length;i++)
            System.out.println(people[i]);
    }

}
```

Quick Quiz

```
Person per1 = new Person("Bob");
Person per2 = new Student("Bob", 1);
Student stu1 = new Student("Bob", 0);
Student stu2 = new Student("Bob", 1);


System.out.println(per1.equals(per2));
System.out.println(per1.equals(stu1));
System.out.println(per1.equals(stu2));


System.out.println(per2.equals(per1));
System.out.println(per2.equals(stu1));
System.out.println(per2.equals(stu2));


System.out.println(stu1.equals(per1));
System.out.println(stu1.equals(per2));
System.out.println(stu1.equals(stu2));

System.out.println(stu2.equals(per1));
System.out.println(stu2.equals(per2));
System.out.println(stu2.equals(stu1));
```

- What is the output for per1, per2, stu1, and stu2?

```
public boolean equals(Person aP)
{
    return aP != null &&
            this.name.equals(aP.getName());
}
```

```
public boolean equals(Student aS)
{
    return aS != null &&
            super.equals(aS) &&
            this.id == aS.getID();
}
```

Quick Quiz

```
Person per1 = new Person("Bob");
Person per2 = new Student("Bob", 1);
Student stu1 = new Student("Bob", 0);
Student stu2 = new Student("Bob", 1);


System.out.println(per1.equals(per2));
System.out.println(per1.equals(stu1));
System.out.println(per1.equals(stu2));


System.out.println(per2.equals(per1));
System.out.println(per2.equals(stu1));
System.out.println(per2.equals(stu2));


System.out.println(stu1.equals(per1));
System.out.println(stu1.equals(per2));
System.out.println(stu1.equals(stu2));

System.out.println(stu2.equals(per1));
System.out.println(stu2.equals(per2));
System.out.println(stu2.equals(stu1));
```

```
true
true
true

true
true
true

true
true
false

true
true
false
```

```
Person per1 = new Person("Bob");
Person per2 = new Student("Bob", 1);
Student stu1 = new Student("Bob", 0);
Student stu2 = new Student("Bob", 1);


System.out.println(per1.equals(per2));
System.out.println(per1.equals(stu1));
System.out.println(per1.equals(stu2));


System.out.println(per2.equals(per1));
System.out.println(per2.equals(stu1));
System.out.println(per2.equals(stu2));


System.out.println(stu1.equals(per1));
System.out.println(stu1.equals(per2));
System.out.println(stu1.equals(stu2));

System.out.println(stu2.equals(per1));
System.out.println(stu2.equals(per2));
System.out.println(stu2.equals(stu1));
```

Quick Quiz

- What is the output for per1, per2, stu1, and stu2?

```
public boolean equals(Person aP)
{
    return aP != null &&
            this.name.equals(aP.getName());
}
```

```
public boolean equals(Person person) {
    if (person instanceof Student) {
        return super.equals(person) && (this.id ==
        ((Student) person).id);
    } else {
        return super.equals(person);
    }
}
```

Quick Quiz

```
Person per1 = new Person("Bob");
Person per2 = new Student("Bob", 1);
Student stu1 = new Student("Bob", 0);
Student stu2 = new Student("Bob", 1);


System.out.println(per1.equals(per2));
System.out.println(per1.equals(stu1));
System.out.println(per1.equals(stu2));


System.out.println(per2.equals(per1));
System.out.println(per2.equals(stu1));
System.out.println(per2.equals(stu2));


System.out.println(stu1.equals(per1));
System.out.println(stu1.equals(per2));
System.out.println(stu1.equals(stu2));

System.out.println(stu2.equals(per1));
System.out.println(stu2.equals(per2));
System.out.println(stu2.equals(stu1));
```

```
true
true
true

true
false
true

true
false
false

true
true
false
```

```
Person per1 = new Person("Bob");
Person per2 = new Student("Bob", 1);
Student stu1 = new Student("Bob", 0);
Student stu2 = new Student("Bob", 1);


System.out.println(per1.equals(per2));
System.out.println(per1.equals(stu1));
System.out.println(per1.equals(stu2));


System.out.println(per2.equals(per1));
System.out.println(per2.equals(stu1));
System.out.println(per2.equals(stu2));


System.out.println(stu1.equals(per1));
System.out.println(stu1.equals(per2));
System.out.println(stu1.equals(stu2));

System.out.println(stu2.equals(per1));
System.out.println(stu2.equals(per2));
System.out.println(stu2.equals(stu1));
```

Quick Quiz

- What is the output for per1, per2, stu1, and stu2?

```
public boolean equals(Person aP)
{
    return aP != null &&
            this.name.equals(aP.getName());
}
```

```
public boolean equals(Object person) {
    if (person instanceof Student) {
        return super.equals(person) && (this.id == ((Student)
        person).id);
    } else if (person instanceof Person) {
        return super.equals(person);
    }
    return false;
}
```

Quick Quiz

```
Person per1 = new Person("Bob");
Person per2 = new Student("Bob", 1);
Student stu1 = new Student("Bob", 0);
Student stu2 = new Student("Bob", 1);


System.out.println(per1.equals(per2));
System.out.println(per1.equals(stu1));
System.out.println(per1.equals(stu2));


System.out.println(per2.equals(per1));
System.out.println(per2.equals(stu1));
System.out.println(per2.equals(stu2));


System.out.println(stu1.equals(per1));
System.out.println(stu1.equals(per2));
System.out.println(stu1.equals(stu2));

System.out.println(stu2.equals(per1));
System.out.println(stu2.equals(per2));
System.out.println(stu2.equals(stu1));
```

```
true
true
true

true
true
true

true
true
true

true
true
true
```

```
Person per1 = new Person("Bob");
Person per2 = new Student("Bob", 1);
Student stu1 = new Student("Bob", 0);
Student stu2 = new Student("Bob", 1);


System.out.println(per1.equals(per2));
System.out.println(per1.equals(stu1));
System.out.println(per1.equals(stu2));


System.out.println(per2.equals(per1));
System.out.println(per2.equals(stu1));
System.out.println(per2.equals(stu2));


System.out.println(stu1.equals(per1));
System.out.println(stu1.equals(per2));
System.out.println(stu1.equals(stu2));

System.out.println(stu2.equals(per1));
System.out.println(stu2.equals(per2));
System.out.println(stu2.equals(stu1));
```

Quick Quiz

- What is the output for per1, per2, stu1, and stu2?

```
public boolean equals(Object per) {
     if (per instanceof Person) {
          return this.name == ((Person) per).name;
     }
return false;
}
```

```
public boolean equals(Object person) {
     if (person instanceof Student) {
          return super.equals(person) && (this.id == ((Student)
          person).id);
     } else if (person instanceof Person) {
          return super.equals(person);
     }
     return false;
}
```

Quick Quiz

```
Person per1 = new Person("Bob");
Person per2 = new Student("Bob", 1);
Student stu1 = new Student("Bob", 0);
Student stu2 = new Student("Bob", 1);


System.out.println(per1.equals(per2));
System.out.println(per1.equals(stu1));
System.out.println(per1.equals(stu2));


System.out.println(per2.equals(per1));
System.out.println(per2.equals(stu1));
System.out.println(per2.equals(stu2));


System.out.println(stu1.equals(per1));
System.out.println(stu1.equals(per2));
System.out.println(stu1.equals(stu2));

System.out.println(stu2.equals(per1));
System.out.println(stu2.equals(per2));
System.out.println(stu2.equals(stu1));
```

```
true
true
true

true
false
true

true
false
false

true
true
false
```