# Computer Science II
# Handout 6

# Inheritance

- Another way for classes/objects to work together

- Allows classes to share certain members
  - You may *extend* an existing class
  - The new class then *inherits* the members of the class it extends

- Can be described as a “*is a*” relationship
  - Contrast with the “*has a*” relationship of aggregation

# Inheritance: the “is a” relationship

- Inheritance is another way that OOP models the real world

- Many characteristics are shared between different classes
  - Apples may be Red Delicious, Granny Smith, Fuji, Macintosh, Honeycrisp, Gala, …
  - An Insect may be a Grasshopper, Bee, Ant, Mosquito, Moth, …
  - A Person may be a Student, Instructor, Dentist, Mountain Climber, Star Wars Fan, …

- The shared characteristics make up the inherited class
- Specific characteristics then distinguish the other classes

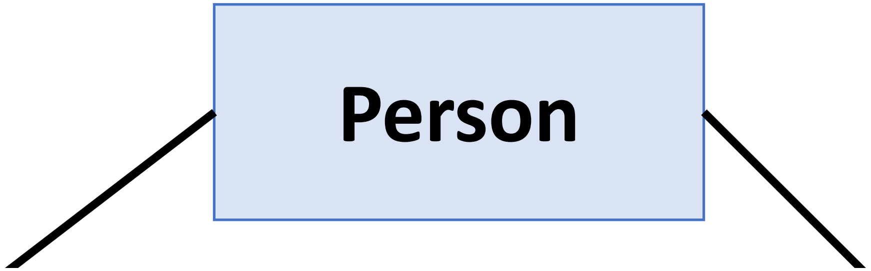
# Inheritance: the “is a” relationship

- Suppose we want to model the hierarchy of the following classes:
  Person, Employee, Faculty, Student, Staff, Undergrad, Grad

- Each of these shares certain characteristics with some of the others

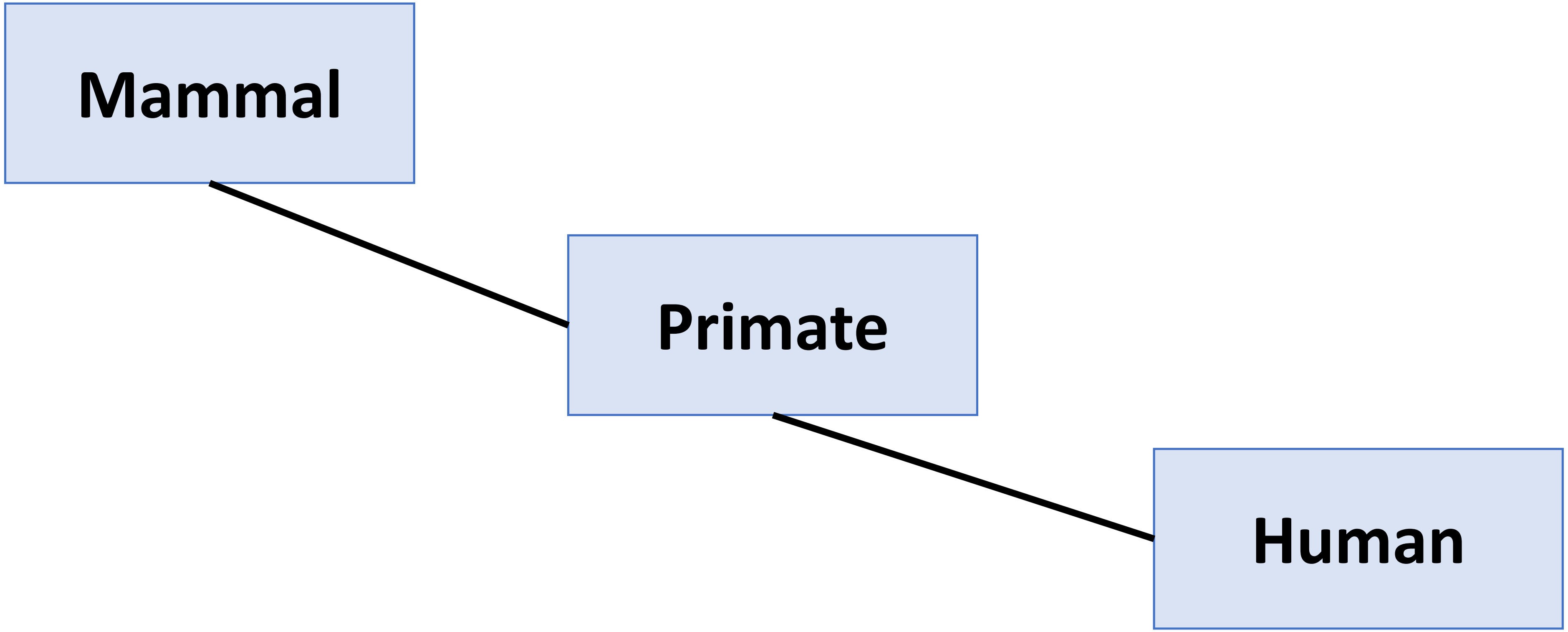
- How would these look when arranged in a hierarchy of classes?

# Inheritance: the “is a” relationship



# Inheritance: the “is a” relationship

- Any classification that can be described using a variant of the phrase “*is a*” can have a similar hierarchy



# Inheritance: the “is a” relationship

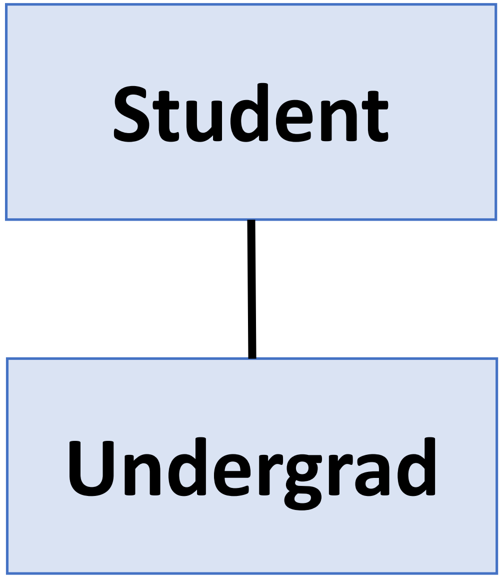
- In this kind of relationship, the more specialized class has all the characteristics of the more general class
  - A human *is a* mammal, so we give birth to live young, have hair, etc.
  - An elephant *is a* mammal, so they give birth to live young, have hair, etc.

  - A human *is a* primate, so we have stereo vision, larger brains, etc.
  - A gorilla *is a* primate, so they have stereo vision, larger brains, etc.

- The more specialized class will also have its own characteristics
  - A human *is a* mammal and *is a* primate and can use the internet

# Inheritance: the “is a” relationship

- When designing your own classes, inheritance allows you to extend the capabilities of one class into another



**Student.java**

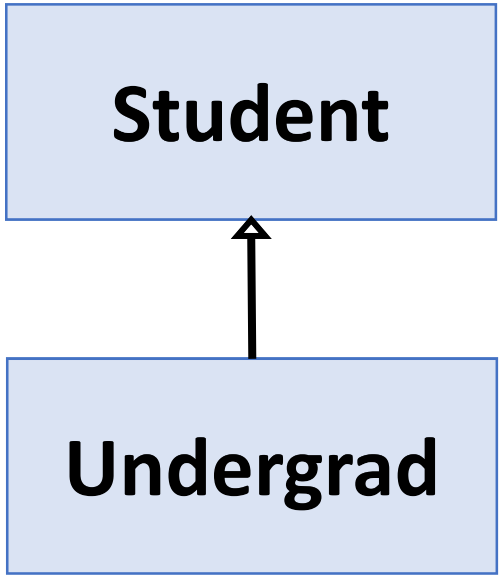
```
public class Student {
        …
}
```

**Undergrad.java**

```
public class Undergrad extends Student {
        …
}
```

# Inheritance: the “is a” relationship

- In a UML diagram, inheritance is indicated with a triangle-shaped arrow



This distinguishes between the *superclass* …
(the generalized class)

… and the *subclass.*

(the specialized class)

# Inheritance: the “is a” relationship

- The subclass *extends* the superclass

- The subclass will inherit attributes and methods from the superclass “for free”
  - Additional attributes and methods then specialize the subclass

- Using inheritance allows you to take advantage of similarities between classes
  - Keeps your code modular
  - Re-uses code without re-writing it

# Inheritance is *not* aggregation

Note how these differ:

- Aggregation uses the "*has a*" relationship
- Inheritance uses the "*is a*" relationship

An Undergrad *has a* Course. An Undergrad *is a* Student.

- Aggregation uses another class by creating an instance of it
- Inheritance uses another class by becoming an instance of it

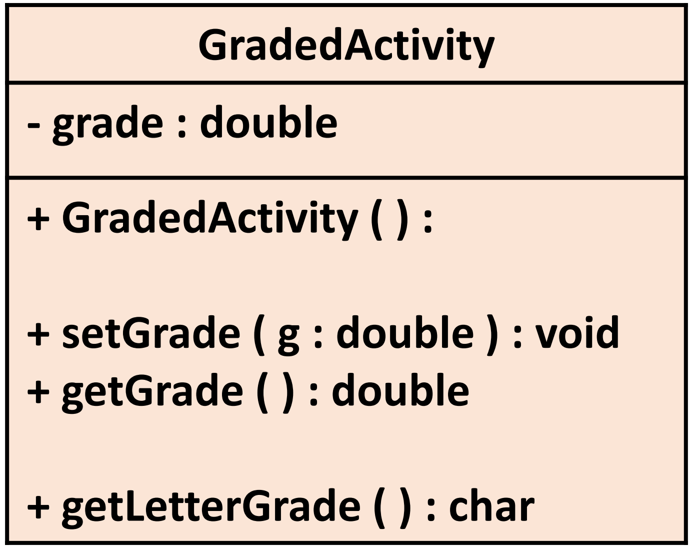
# Inheritance is *not* aggregation

Note how these differ:

- Deciding between the two means deciding on what kind of relationship your classes should have

# Inheritance – example

- Design a **`GradedActivity`** class that will hold the numerical grade of a student in some activity (a quiz, a test, an assignment, etc.) and calculate the letter grade



**GradedActivity**

- grade : double

+ GradedActivity ( ) :

+ setGrade ( g : double ) : void
+ getGrade ( ) : double

+ getLetterGrade ( ) : char

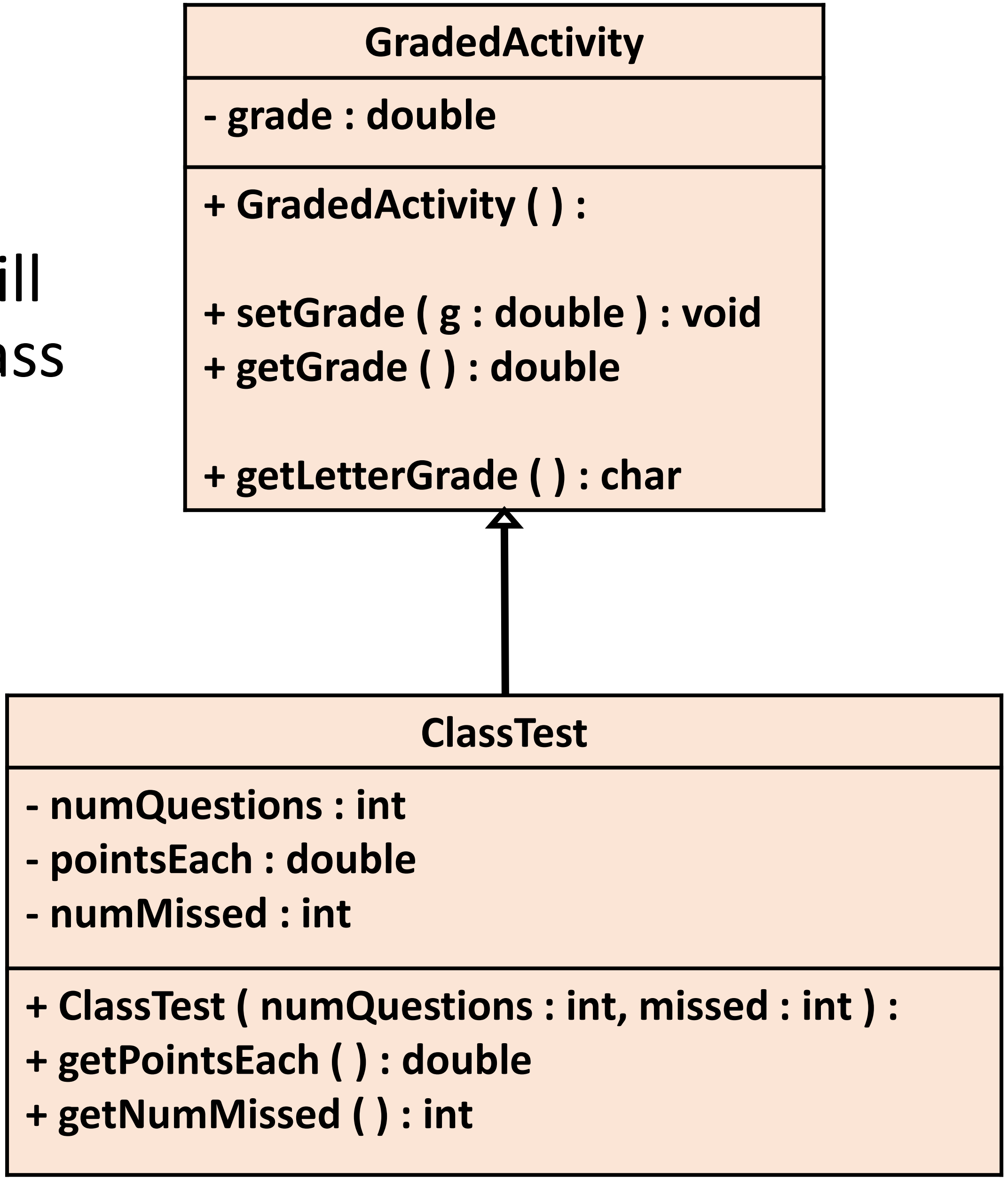
```
public class GradedActivity {
        private double grade;

        // Constructor
        public GradedActivity () { }

        public void setGrade(double g) {
                grade = g;
        }

        public double getGrade() {
                return grade;
        }

        public char getLetterGrade() {
                if(score >= 90)
                        return ‘A’;
                else if(score >= 80)
                        return ‘B’;
                else if(score >= 70)
                        return ‘C’;
                else if(score >= 60)
                        return ‘D’;
                else
                        return ‘F’;
        }
}
```

```
import java.util.Scanner;
public class GradeDemo {
      public static void main(String[] args) {
            GradedActivity activity = new GradedActivity();

            Scanner kb = new Scanner(System.in);
            System.out.print(“Enter numeric score (0-100): ”);

            double mark = kb.nextDouble();
            activity.setGrade(mark);

            System.out.println(“Grade is ” + activity.getGrade());

      }
}
```

```
> Enter numeric score (0-100): 73
```

# Inheritance – example

- We can now create another class that will inherit from the **`GradedActivity`** class



GradedActivity

- grade : double

+ GradedActivity ( ) :

+ setGrade ( g : double ) : void
+ getGrade ( ) : double

+ getLetterGrade ( ) : char

ClassTest

- numQuestions : int
- pointsEach : double
- numMissed : int

+ ClassTest ( numQuestions : int, missed : int ) :
+ getPointsEach ( ) : double
+ getNumMissed ( ) : int

```java
public class ClassTest extends GradedActivity {
        private int numQuestions;
        private double pointsEach;
        private int numMissed;


        // Constructor
        public ClassTest (int q, int m) {
                double grade;
                numQuestions = q;
                numMissed = m;
                pointsEach = 100.0 / numQuestions;


                setGrade(grade); // calls the superclass method!
        }


        public double getPointsEach() {
                return pointsEach;
        }


        public int getNumMissed() {
                return numMissed;
        }
}
```

```
import java.util.Scanner;
public class ClassTestDemo {
        public static void main(String[] args) {
                Scanner kb = new Scanner(System.in);
                System.out.print(“Number of questions: ”);
                int q = kb.nextInt();
                System.out.print(“Number missed: ”);
                int m = kb.nextInt();

                ClassTest test1 = new ClassTest(q, m);

                System.out.println(“Grade: ” + test1.getGrade());
                System.out.println(“Letter grade: ” + test1.getLetterGrade());
                // Both of the above calls come from GradedActivity


        }

}
```

```
> Number of questions: 50
  Number missed: 4
```

# Inheritance

- **`ClassTest`** was the *subclass* and **`GradedActivity`** was the *superclass*
  - So **`ClassTest`** inherited members from **`GradedActivity`**

- We did not need to write the inherited methods/variables!

**Variables**
**from `ClassTest`**

numQuestions
pointsEach
numMissed

**Methods**
**from `ClassTest`**

Constructor
getPointsEach
getNumMissed

**Methods**
**from `GradedActivity`**

setGrade
getGrade
getLetterGrade

# Inheritance

- Note that *private* members were not inherited
  - Though they are “there” in memory, and can still be accessed using proper getter/setter methods!

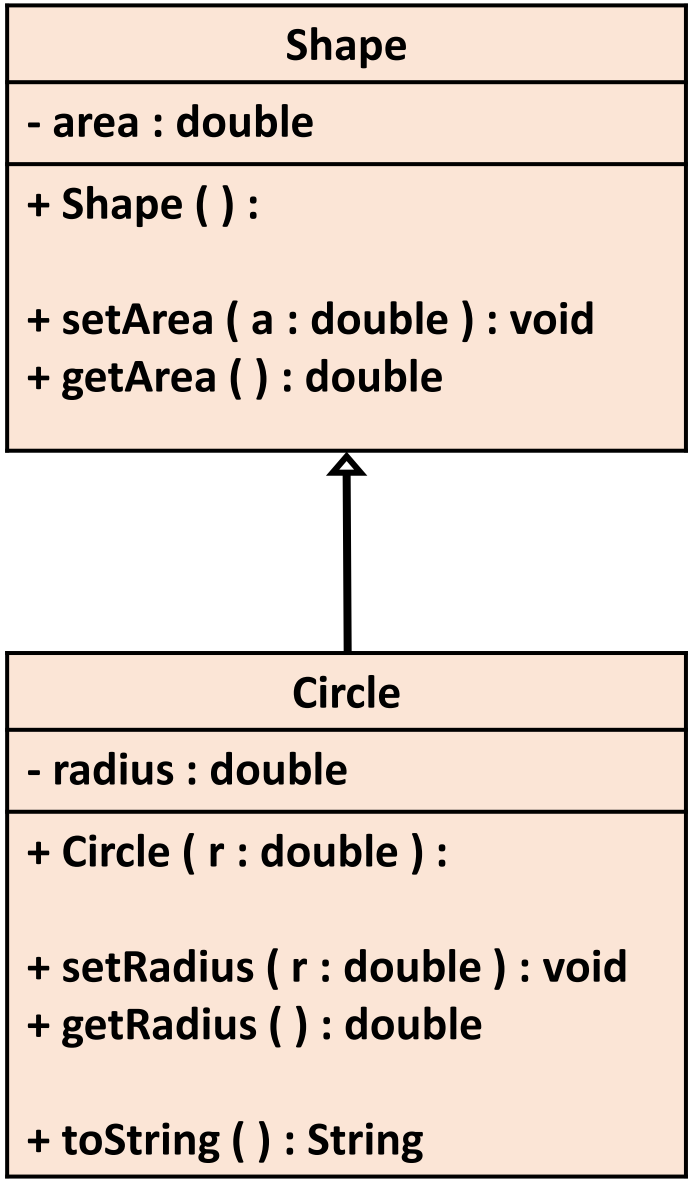
- Note that inheritance does not work in reverse!
  - The **`GradedActivity`** class does not inherit anything from the **`ClassTest`** class

- We can repeat this process: extending **`ClassTest`** to class **`FinalExam`** would create a more specialized case
  - This would inherit everything from **`ClassTest`** and **`GradedActivity`**

# Inheritance – example 2

- Design a Shape class that has
  - an instance variable for its area
  - a default constructor
  - get/set methods

- Use this as the superclass and extend it to the Circle class that has:
  - an instance variable for its radius
  - a constructor that sets the radius and area
  - get/set methods
  - a toString method that prints the radius and area



**Shape**

- area : double

+ Shape ( ) :

+ setArea ( a : double ) : void
+ getArea ( ) : double

**Circle**

- radius : double

+ Circle ( r : double ) :

+ setRadius ( r : double ) : void
+ getRadius ( ) : double

+ toString ( ) : String

```java
public class Shape {
      private double area;

      public Shape () { }

      public double getArea() {
            return area;
      }

      public void setArea(double a) {
            area = a;
      }
}
```

```
public class Circle extends Shape {
        private double radius;

        public Circle (double r) {
                setRadius(r);  // calls our own set method!
        }

        public void setRadius(double r) {



        }

        public double getRadius() {
                return radius;
        }

        public String toString() {
                return “Radius=” + radius + “\nArea=” + getArea();
        }
}
```

```
import java.util.Scanner;
public class CircleDemo {
      public static void main(String[] args) {
            Scanner kb = new Scanner(System.in);
            System.out.print(“Radius? ”);
            double r = kb.nextDouble();

            Circle c = new Circle(r);

            System.out.println(c);



      }

}
```

```
> Radius? 5
  Radius: 5.0
  Area: 78.53981633974483
```