# Inheritance

Colorado State University

Department of Computer Science

Slides Originally Created by Albert Lionelle (Albert.Lionelle@colostate.edu),
updated by Marcia Moraes (marcia.moraes@colostate.edu)

# Announcements

**NEXT WEEK**
**Exam 2**

Don’t procrastinate and catch up if you need!

“DON'T SIT DOWN AND WAIT FOR THE OPPORTUNITIES TO COME. GET UP AND MAKE THEM.”

MADAM C.J. WALKER

https://www.ellevatenetwork.com/articles/8013-inspirational-quotes-from-black-women-pioneers

TODO Reminders:

Readings are due **before** lecture

- Reading 14 (zybooks) – you should have already done that ☺
- Lab 09 – go to your lab to have your participation points
- Reading 15 (zyBooks) – you should have already done that ☺
- Lab 10 – go to your lab to have your participation points
- Reading 16 (zybooks) – you should have already done that ☺

- RPA 7 – remember to do it by Sunday or earlier to have your Exam’s module open!

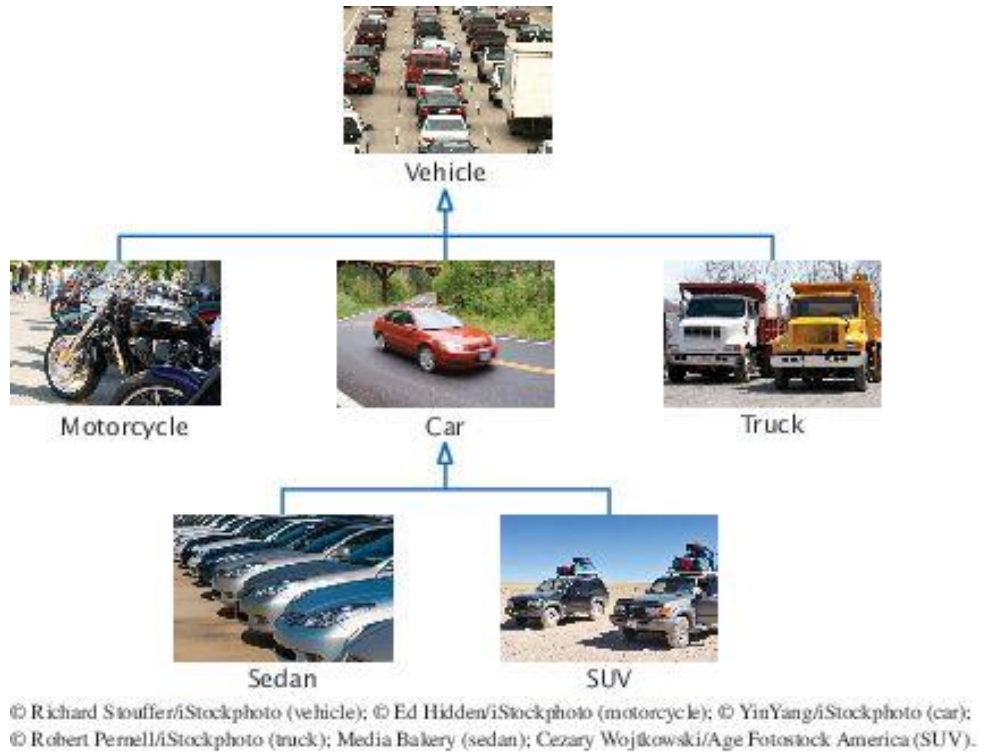
Keep practicing your RPAs in a spaced and mixed manner ☺

Help Desk

| Day | Time : Room |
|---|---|
| Monday | 12 PM - 2 PM : CSB 120 |
| Tuesday | 6 PM - 8 PM : Teams |
| Wednesday | 3 PM - 5 PM : CSB 120 |
| Thursday | 6 PM - 8 PM : Teams |
| Friday | 3 PM - 5 PM : CSB 120 |
| Saturday | 12 PM - 4 PM : Teams |
| Sunday | 12 PM - 4 PM : Teams |

# Inheritance

- Is a relationship between a more general class (called **superclass**) and a more specialized class (called **subclass**)
- The subclass inherits data and behavior from the superclass

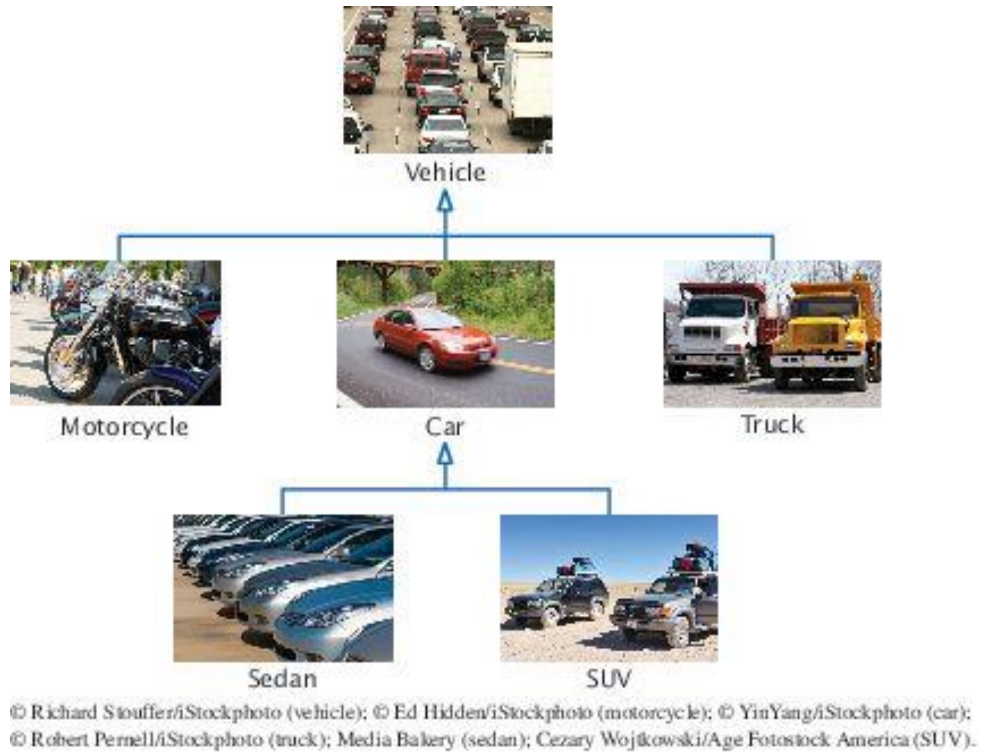

Vehicle
Motorcycle
Car
Truck
Sedan
SUV
© Richard Stouffer/iStockphoto (vehicle); © Ed Hidden/iStockphoto (motorcycle); © YinYang/iStockphoto (car);
© Robert Pernell/iStockphoto (truck); Media Bakery (sedan); Cezary Wojtkowski/Age Fotostock America (SUV).

Horstmann, C. (2013) Java for Everyone: Late Objects. Chapter 9, Figure 1, p. 416.

1. Car is a vehicle.
2. The class Car inherits from class Vehicle.
3. In this relationship Vehicle is the superclass and Car is the subclass.
4. Superclass and subclass are joined with an arrow that point to superclass.

# Inheritance – Substitution Principle

- Substitution principle states that you can always use a subclass object when a superclass object is expected.
- What does that mean in practice? Let's consider our Vehicle hierarchy of classes



Vehicle
Motorcycle
Car
Truck
Sedan
SUV
© Richard Stouffer/iStockphoto (vehicle); © Ed Hidden/iStockphoto (motorcycle); © YinYang/iStockphoto (car);
© Robert Pernell/iStockphoto (truck); Media Bakery (sedan); Cezary Wojtkowski/Age Fotostock America (SUV).

Horstmann, C. (2013) Java for Everyone: Late Objects. Chapter 9, Figure 1, p. 416.

1. Consider a method that takes an argument type Vehicle

```
void processVehicle(Vehicle v)
```

2. Because Car is a subclass of Vehicle, you can call the method with a Car object:

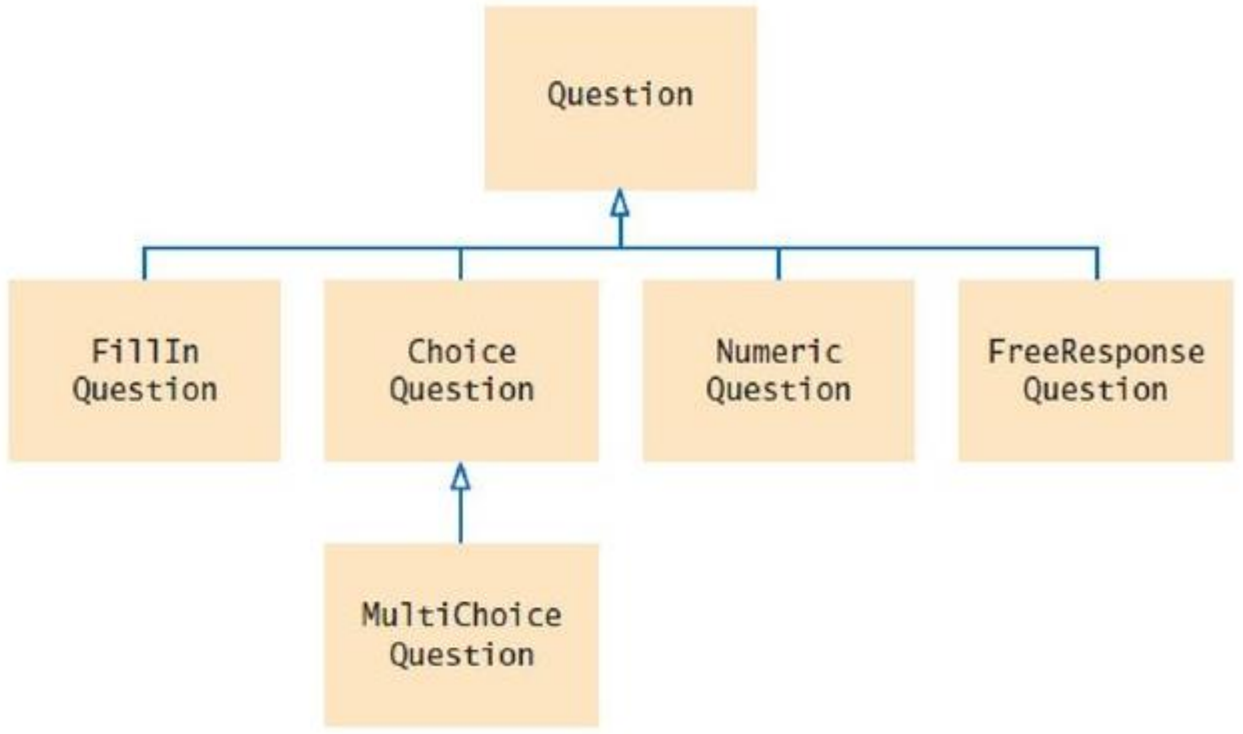
```
Car car1 = new Car( … );
processVehicle(car1);
```

3. Why provide a method that processes Vehicle objects instead of Car objects?

That method is more useful because it handles ANY kind of vehicles (Car, Truck, and Motorcycle)!

# Inheritance – Example

- Consider the following hierarchy of question types
- The root of this hierarchy is the Question type



Horstmann, C. (2013) Java for Everyone: Late Objects. Chapter 9, Figure 3, p. 417.

- Considering that all question types can display its text and can check whether a given response is a correct answer, which class should implement these functionalities?

  Question

- What data and behaviors should be implemented?

  Data
  - text and answer

  Behavior
  - constructor
  - set
  - get
  - checkAnswer
  - display

# Inheritance – Example

```
public class Question {
   private String text;
   private String answer;
   public Question() {
      //calls the constructor with two parameters
      this("", "");
   }
   public Question(String text, String answer) {
      setText(text);
      setAnswer(answer);
   }
   public void setText(String text) {
      this.text = text;
   }
   public void setAnswer(String answer) {
      this.answer = answer;
   }
   public String getText() {
      return text;
   }
   public String getAnswer() {
      return answer;
   }
   public boolean checkAnswer(String answer){
      return (this.answer.equals(answer));
   }
   public void display(){
      System.out.println(text);
   }
}
```

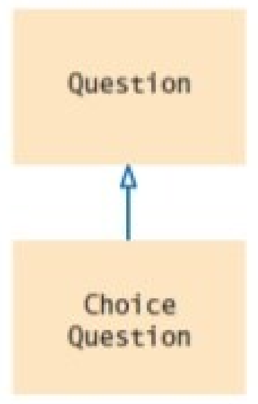
```
import java.util.Scanner;
public class QuestionApp {
   public static void main(String args[]){
      Scanner in = new Scanner(System.in);
      Question q = new Question("Who was the inventor of Java?", "James Gosling");
      q.display();
      System.out.println("You answer: ");
      String response = in.nextLine();
      System.out.println(q.checkAnswer(response));
   }
}
```

# Inheritance – Example

- Suppose we want our program to handle questions like this:

  In which country was the inventor of Java born?
  1. Australia
  2. Canada
  3. Denmark
  4. United States

- What we could do?



Use inheritance to implement ChoiceQuestion
as a subclass of Question!

Horstmann, C. (2013) Java for Everyone: Late Objects. Chapter 9, Figure 4, p. 420.

- Subclass
  - Automatically have the instance variables that are declared in the superclass
  - Inherits all public methods from the superclass
- We declare new instance variables (attributes) in the subclass
- We declare new methods (behaviors) in the subclass
- We can change the implementation of inherited methods if the inherited behavior is not appropriated – this is called **override**

## Inheritance – Example

- Considering the ChoiceQuestion format below:

  In which country was the inventor of Java born?
  1. Australia
  2. Canada
  3. Denmark
  4. United States
- What instance variables and methods do we need to have on ChoiceQuestion class?

Instance variable

- ArrayList of Strings to store various choices for answers

Methods

- addChoice(String choice, Boolean correct) – new method
- display() – override display method from the superclass

- Okay, so how does the constructor on a subclass works?
- We need to initialize the instance variables that are inherited as well the new instance variables (those who belong to the subclass)
- To initialize the inherited instance variables
  - We do that by calling the constructor of the super class using the reserved work **super** and passing the necessary parameters
- To initialize the new instance variables
  - We do what we always have done so far ☺

# Inheritance – Example

```java
import java.util.ArrayList;
public class ChoiceQuestion extends Question {
   private ArrayList<String> choices;

   public ChoiceQuestion(String questionText){
      super(questionText, "");
      choices = new ArrayList<String>();
   }
   public void addChoice(String choice, boolean correct){
      choices.add(choice);
      if(correct){
         setAnswer(choices.get(choices.size()-1));
      }
   }
   public void display(){
      //display the question text
       super.display();
       //display the answer choices
       for(int i = 0; i < choices.size(); i++) {
          int choiceNumber = i + 1;
          System.out.println(choiceNumber + ": " + choices.get(i));
       }
   }
}
```

extends denotes inheritance

declaring instance variable added to subclass

super – calls the super class constructor

initializing instance variables

new method added to the subclass

using the inherited method setAnswer

overriding method display

calling the super class method display

# Inheritance – Example

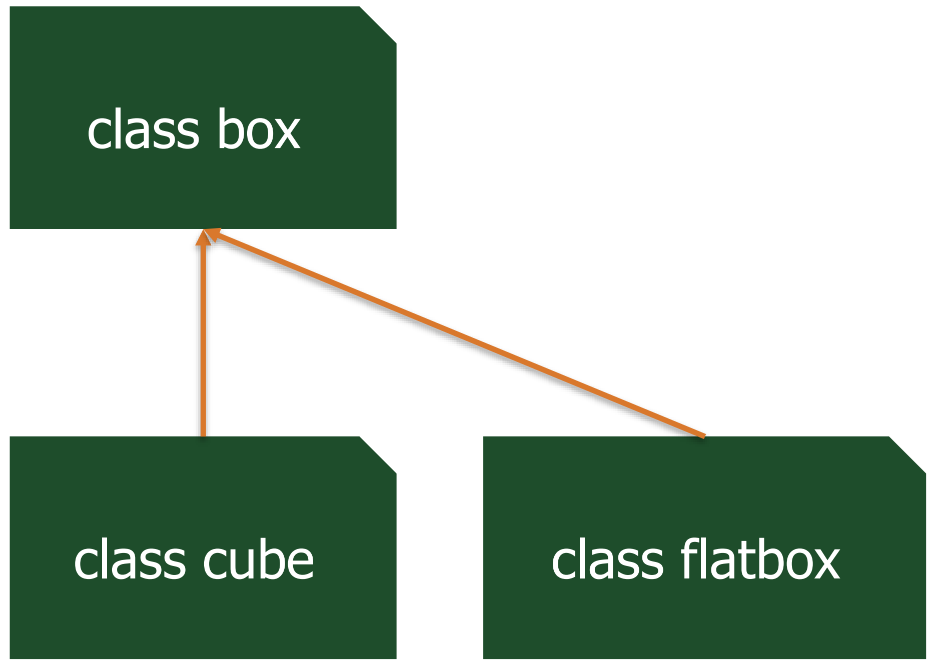
```
import java.util.Scanner;
public class QuestionApp2 {
   public static void main(String args[]) {
      ChoiceQuestion first = new ChoiceQuestion("What was the original name of the Java language?");
      first.addChoice("*7", false);
      first.addChoice("Duke", false);
      first.addChoice("Oak", true);
      first.addChoice("Gosling", false);
      ChoiceQuestion second = new ChoiceQuestion("In which country was the inventor of Java born?");
      second.addChoice("Australia", false);
      second.addChoice("Canada", true);
      second.addChoice("Denmark", false);
      second.addChoice("United States", false);
      presentQuestion(first);
      presentQuestion(second);
   }
```

```
public static void presentQuestion(ChoiceQuestion q) {
      Scanner in = new Scanner(System.in);
      q.display();
      System.out.println("You answer: ");
      String response = in.nextLine();
      System.out.println(q.checkAnswer(response));
   }
}
```

## Inheritance Summary and More Examples

# Inheritance – Makes Java DRY

- Inheritance
  - Heart of OOP!
  - Essential to large programs
  - DRY
- A class can **extend** another class
  - By extending:
    - **inherit** methods and properties!
  - **override**
    - allows you to change methods for children



# Box-Cube Example

Parent

```
public class Box {
    private int width;
    private int height;
    private int length;
    /** getters and setters **/
    public int getArea() {
        return getHeight()*getLength()*getWidth();
    }
    public String toString() {
        return String.format("Width: %d Height: %d Length: %d",
                getWidth(), getHeight(), getLength());
    }
    public Box(int width, int height, int length) {
        setHeight(height);
        setWidth(width);
        setLength(length);
    }
}
```

Child

```
public class Cube extends Box{
    protected int sides;
    public Cube(int side) {
        super(side, side, side);
        this.sides = side;
    }
    @Override
    public String toString() {
       return String.format("Side Length: %d Area: %d",
                            getWidth(), getArea());
    }
}
```

```
public static void main(String[] args) {
    Box bx = new Box(10, 12, 7);
    Cube cb = new Cube(5);
    System.out.println(bx);
    System.out.println(cb);
}
```

```
Width: 10 Height: 12 Length: 7
Side Length: 5 Area: 125
```

# Object Class

- **All** classes in java extend Object
- **Object** is a type / class
  - Includes common methods
  - toString()
    - returns String of the object
    - by default memory location (not useful) – should override!
    - System.out.println() – calls toString()
    - String concatenation calls toString()
  - equals(Object)
    - compares memory locations
    - should usually override

Objects are the cells of java

## Revisiting Scope

- public
  - Everyone has access
- private
  - Only the class has access
  - This means child classes – can't access private!
- protected
  - child class has access only
- <blank/omitted>
  - package and children have access

```java
public class SuperCube extends Cube{

 @Override
 public int getArea() {
     return width*width*length*length*height;
 }

}
```

private!

Not Allowed

```java
public class SuperCube extends Cube{

 @Override
 public int getArea() {
     return sides * 5;
 }

}
```

protected!

Allowed

## Inheritance is Polymorphic!

- Substitution principle states that you can always use a subclass object when a superclass object is expected.
- Children may appear to be their parents!
- Define a data structure of the parent type and you can store parent and children types!
- Calls correct class!

```
public static void main(String[] args) {
    Box bx = new Box(10, 12, 7);
    Cube cb = new Cube(5);

    ArrayList<Box> boxes = new ArrayList<>();
    boxes.add(bx);
    boxes.add(cb);
    System.out.println(boxes);
}
```

```
[Width: 10 Height: 12 Length: 7, Side Length: 5 Area: 125]
```

Box toString                Cube toString

- Pretty cool
- Will learn more after Exam 2
- **Take away:**
  - inheritance is DRY
  - inheritance lets **children** use methods from parent