# Chapter 2.

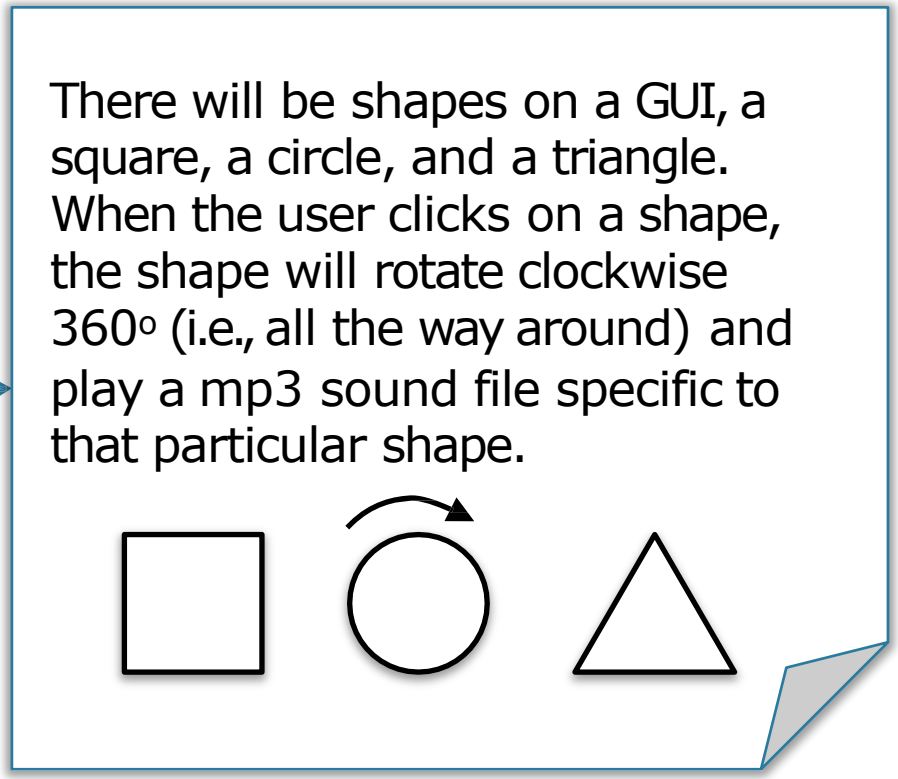

# Class and Object

**2020-2021**
**COMP2396 Object-Oriented Programming and Java**
**Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)**
**Department of Computer Science, The University of Hong Kong**

# A Motivating Example

- Paul, the procedural programming guy, and Ocean, the OO guy, were told to develop a software based on the following specification

the spec



# Procedural Approach

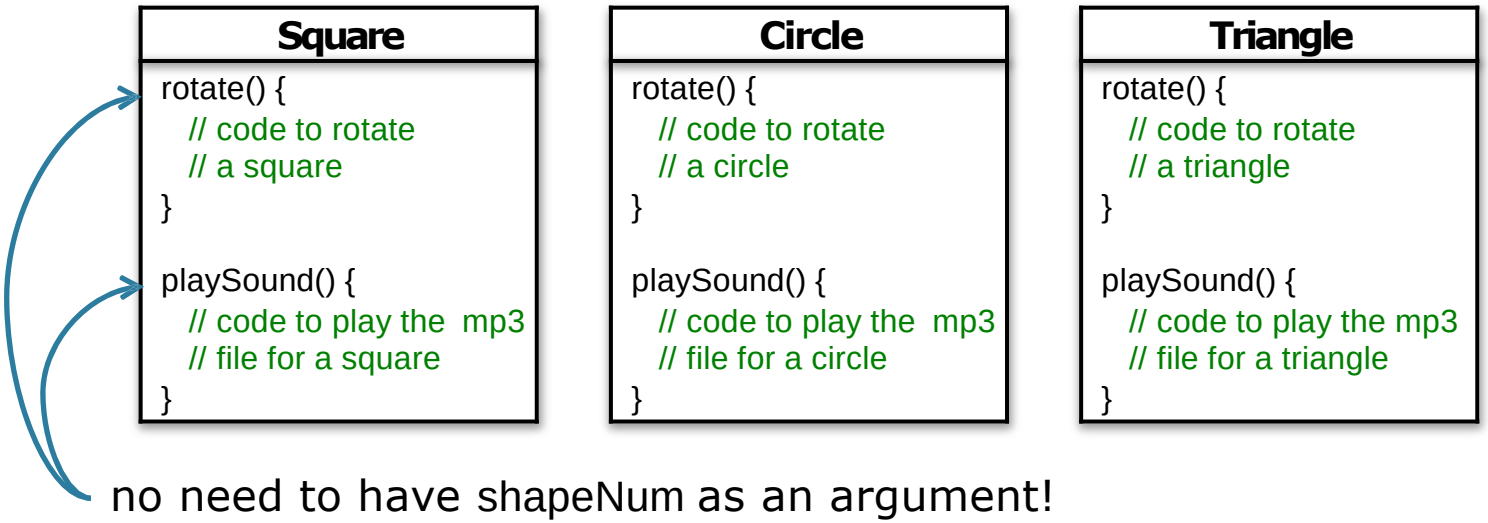
- Paul designed his software by asking himself
  - *What are the tasks this program has to do?*
  - *What procedures do we need?*
- Paul identified and implemented the following 2 procedures

```
rotate(shapeNum) {
   // make the shape rotate 360o
}

playSound(shapeNum) {
   // use shapeNum to lookup which
   // mp3 file to play, and play it
}
```

# Object-Oriented Approach

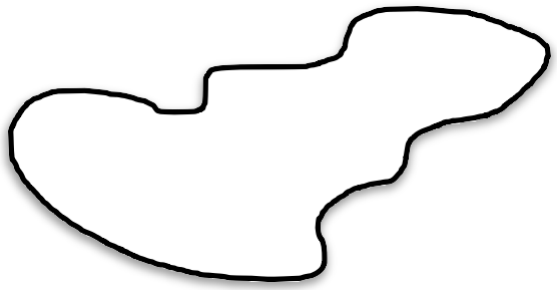
- Ocean designed his software by asking himself
  - *What are the things in this program?*
  - *Who are the key players?*
- Ocean identified the shapes being the key players and wrote a class for each of the 3 shapes



# Spec Change

- A "minor" change was made to the specification

There will be an amoeba shape on the screen, with the others. When the user clicks on the amoeba, it will rotate like others, and play a wav sound file.



What should be added to the spec?

# Spec Change

- Paul needed to update his playSound() procedure

```
playSound(shapeNum) {
   // if the shape is not an amoeba
        // use shapeNum to lookup which
        // mp3 file to play, and play it
   // else
        // play amoeba wav file
}
```

- Ocean just wrote a new class for an amoeba shape

**Amoeba**

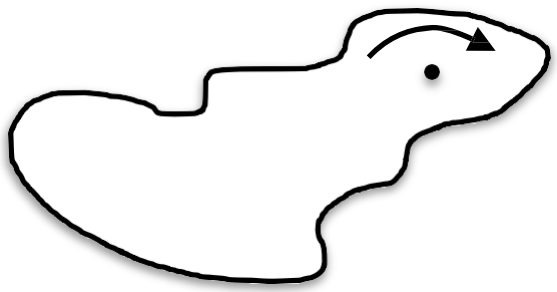
```
rotate() {
   // code to rotate
   // an amoeba
}

playSound() {
   // code to play the wav
   // file for an amoeba
}
```

# Another Spec Change

- Another "minor" change was made to the specification

The amoeba shape should rotate around a specific point instead of the center of its bounding box like the other shapes do.



What should be added to the spec?

# Another Spec Change

- Paul needed to update his `rotate()` procedure and introduce new arguments

```
rotate(shapeNum, x, y) {
   // if the shape is not an amoeba
        // calculate the center point
        // then rotate
   // else
        // use the (x, y) as
        // the point of rotation
        // then rotate
}
```

- Ocean only needed to modify the `rotate()` method of his Amoeba class and add two instance variables

**Amoeba**

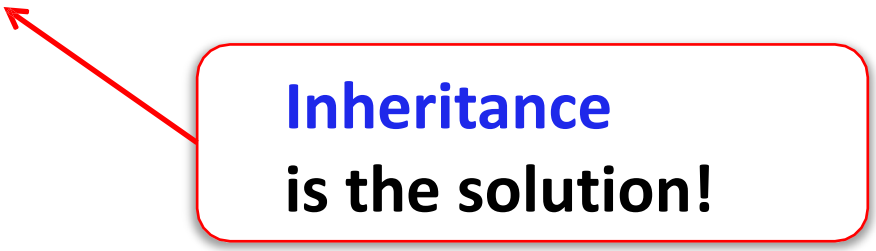
```
int x
int y

rotate() {
   // code to rotate
   // an amoeba
}

playSound() {
   // code to play the wav
   // file for an amoeba
}
```

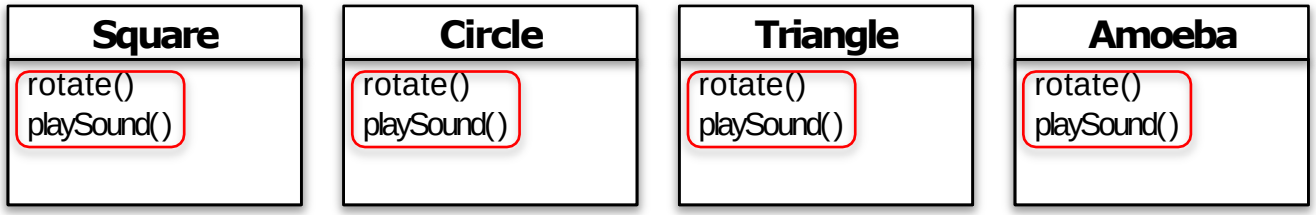
# Procedural vs Object-Oriented

- Paul's code (procedural approach)
  - Making a minor change often involves touching previously tested code
  - Code is difficult to maintain because data and procedures are separated
- Ocean's code (OO approach)
  - Making a modification often does not involve touching previously tested code for other parts of the program
  - Duplicate code (need to maintain 4 different `rotate()` methods)



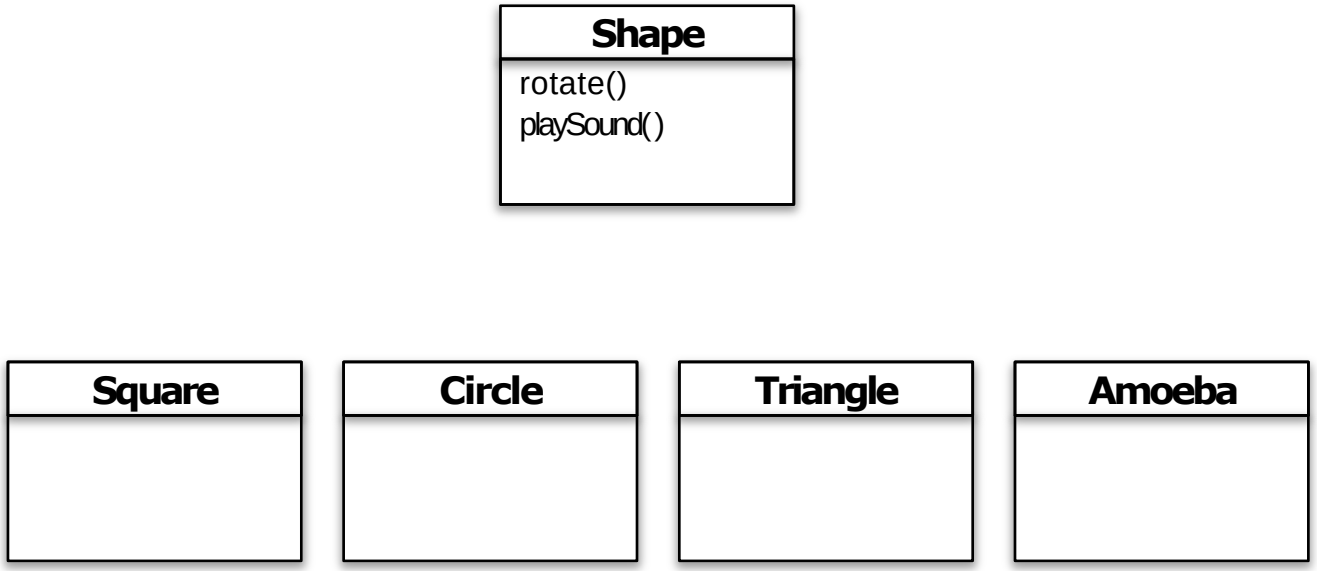
# Inheritance

- Identify the common features from the 4 different shape classes
  - rotate()
  - playSound()



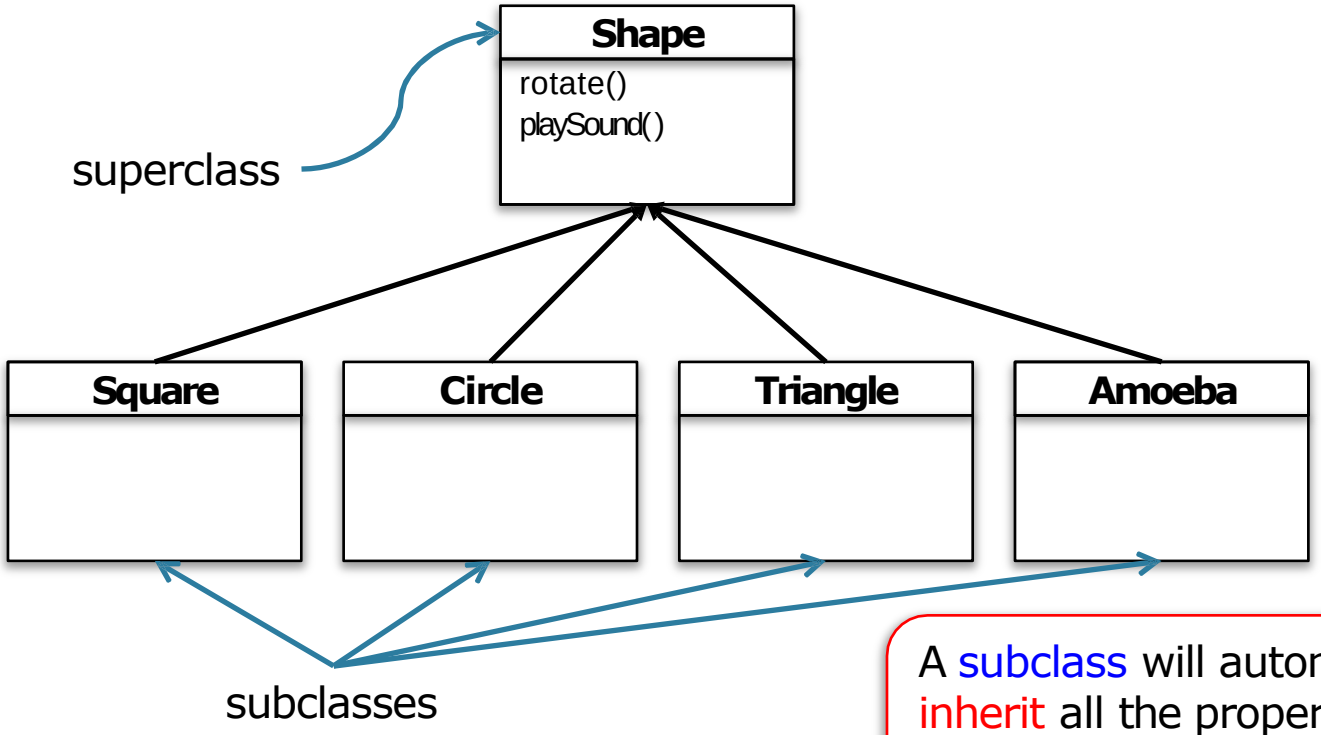
# Inheritance

- Abstract these common features out and put them in a new class called Shape



# Inheritance

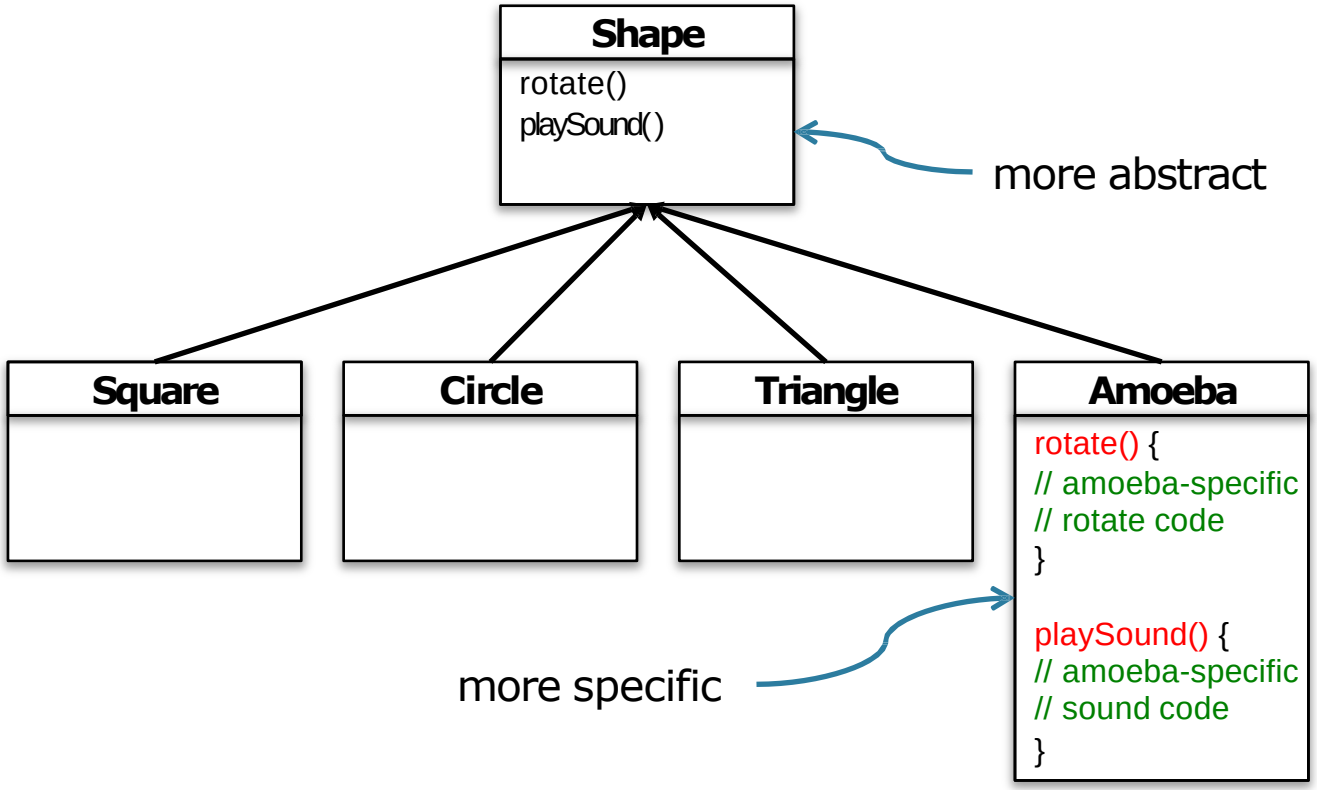
- Make the new Shape class the superclass of the 4 shape classes




A subclass will automatically inherit all the properties (both instance variables and methods) of its superclass

# Inheritance

- Make the Amoeba class override the `rotate()` and playSound() methods of its superclass

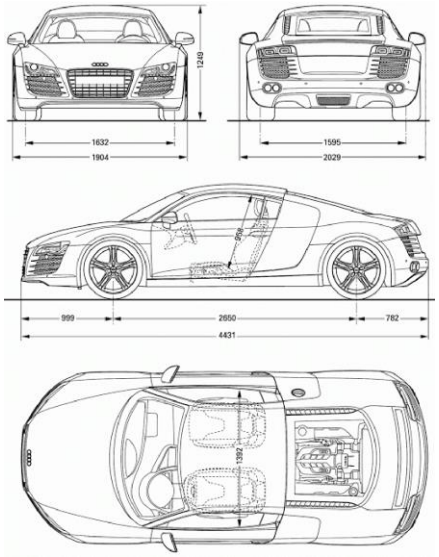


# Some Advantages of OOP

- Modularity
  - The code for an object can be written and maintained independently of the code for other objects
- Information hiding
  - An object's internal implementation remains hidden
- Code re-use
  - The code for an existing object can be reused either directly or through inheritance
- Pluggability
  - Problematic objects can be replaced just like the way mechanical problems are being fixed in the real world

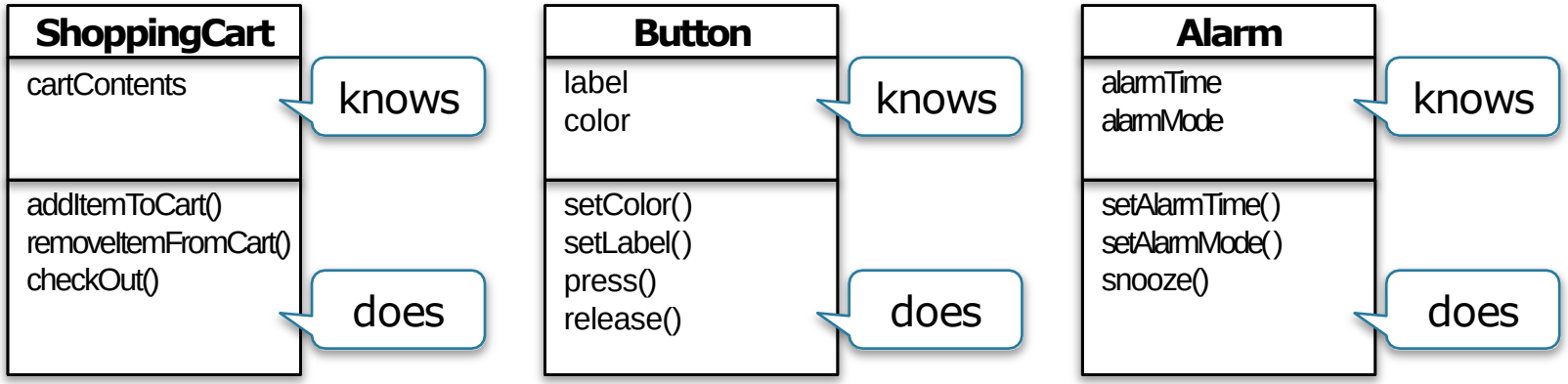
# Classes and Objects

- Classes and objects are the basic building blocks in OOP
- A class is not an object, but a blueprint for an object
- It tells the virtual machine how to make an object of that particular type
- Each object made from that class (i.e., an instance of that class) is unique and has its own state
- Example:
  - The Button class can be used to make dozens of different buttons, and each button might have its own color, size, shape, label, etc.



# Designing a Class

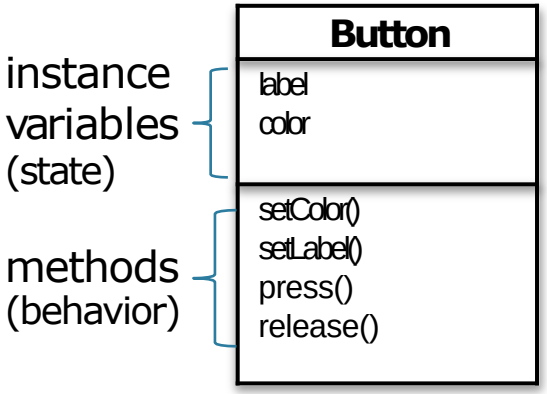
- When designing a class, think about the objects that will be created from that class type
- Objects can be used to model real world objects (e.g., car, dog), as well as concepts (e.g., date, bank account)
- Think about
  - Things the object knows (state)
  - Things the object does (behavior)



# Instance Variables and Methods

- Objects have instance variables and methods, which are designed as part of the class
  - Instance Variables
    - Things an object knows about itself
    - Represent an object's state (data)
    - Can have unique values for each object of that type
  - Methods
    - Things an object can do
    - Define the behavior of an object
    - Operate on data of an object

    (e.g., it is common for an object to have methods that read or write values of its instance variables)



# Example: Tetris

- What are the objects in this game?

| Piece |
| --- |
| |
| |

| Board |
| --- |
| |
| |

# Example: Tetris

- What are their states and behaviors?

instance variables (state)

methods (behavior)

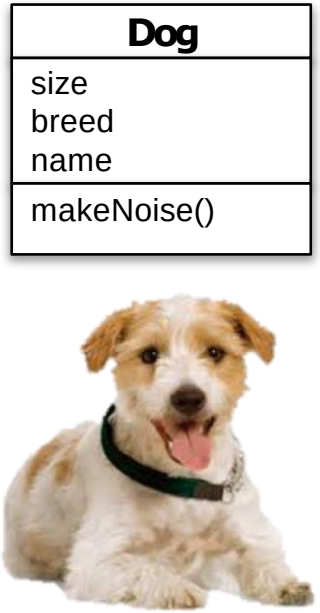
| Piece |
| --- |
| orientation<br>position<br>shape<br>color |
| moveLeft()<br>moveRight()<br>rotate()<br>fall() |

| Board |
| --- |
| level<br>score<br>pieces<br>nextPieces |
| removeRow()<br>levelUp()<br>checkEndOfGame() |

# Defining a Class in Java

- In Java, a class is defined using the keyword class

```
class Dog {
   int size;
   String breed;
   String name;
   void makeNoise() {
      System.out.println("Woof!");
   }
}
```



# Defining a Class in Java

- In Java, a class is defined using the keyword class

defining a new class

name of the class

```
class Dog {
   int size;
   String breed;
   String name;
   void makeNoise() {
      System.out.println("Woof!");
   }
}
```

| Dog |
| --- |
| size<br>breed<br>name |
| makeNoise() |

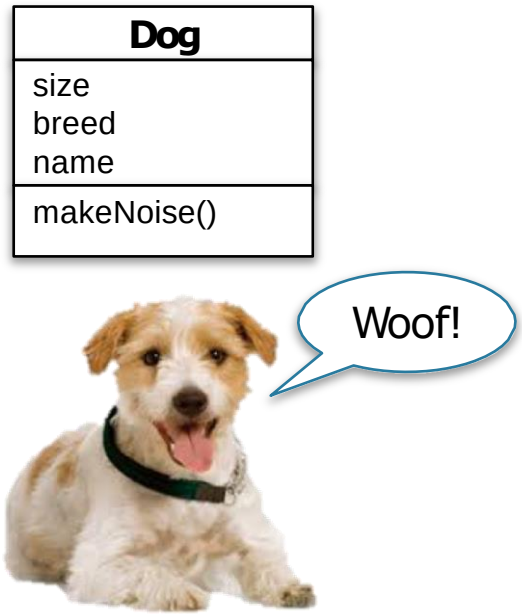
# Defining a Class in Java

- In Java, a class is defined using the keyword class

instance variables

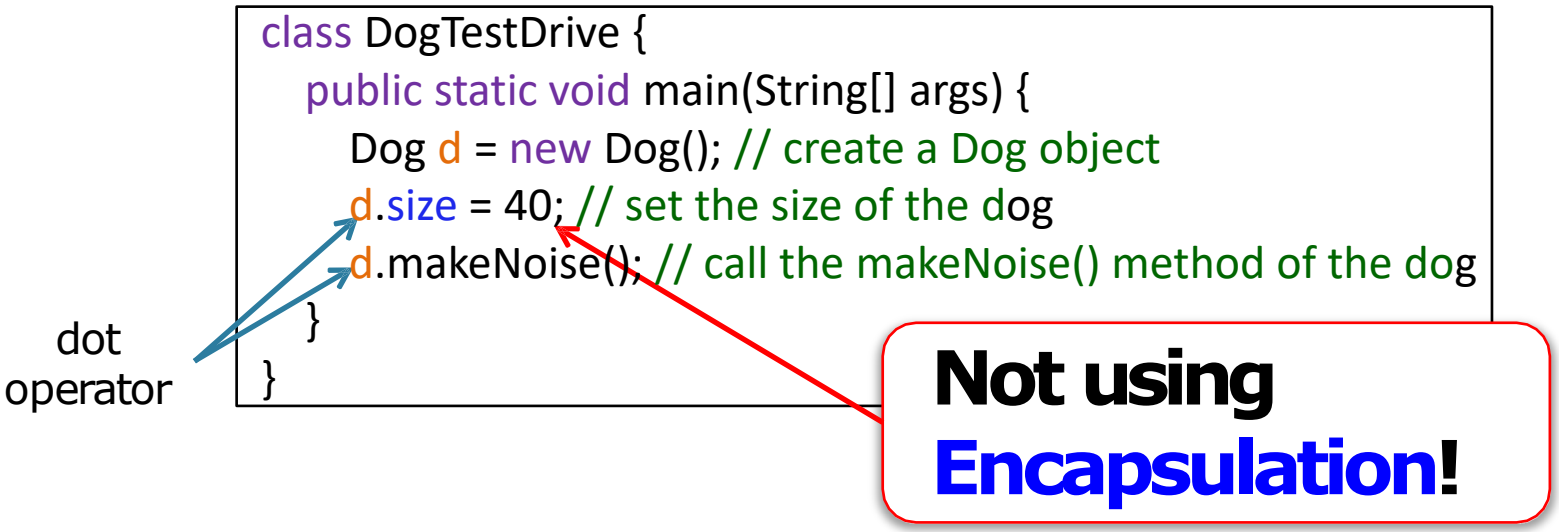
a method

```
class Dog {
   int size;
   String breed;
   String name;
   void makeNoise() {
      System.out.println("Woof!");
   }
}
```



# A Tester Class

- A tester class is often used to test a new class
- Inside the main() method of a tester class
  - Create objects of the new class using the new operator
  - Access the instance variables and methods of the new objects using the dot operator



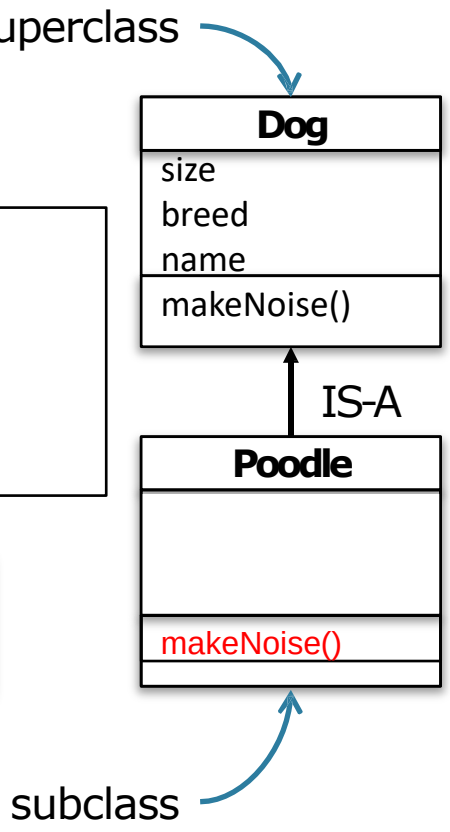
# Inheritance in Java

- In Java, a class (subclass) can be derived from an existing class (superclass) using the keyword extends

overrides the makeNoise() method

```
class Poodle extends Dog {
    void makeNoise() {
        System.out.println("Ruff! Ruff!");
    }
}
```

Poodle IS-A Dog, it will automatically inherit all the instance variables and methods of Dog



# Inheritance in Java

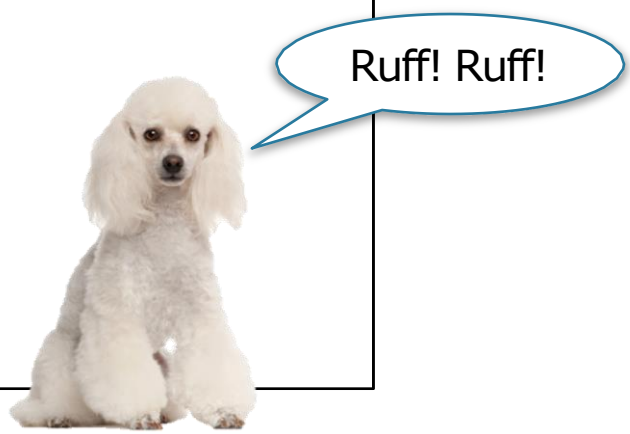
- Example

```
class PoodleTestDrive {
   public static void main(String[] args) {
      Dog d = new Poodle(); // create a Poodle object
      d.size = 25; // set the size of the dog
      d.makeNoise(); // call the makeNoise() method of the dog
   }
}
```

**Polymorphism!**

- Sample output

```
Ruff! Ruff!
```



# Challenge

- Suppose Poodle is modified as follows:

```
class Poodle extends Dog {
    void makeNoise() {
        System.out.println("Ruff! Ruff!");
    }
    void sing() {
        System.out.println("Singing");
    }
}
```

- What is the output of the following?

```
class PoodleTestDrive {
   public static void main(String[] args) {
      Dog d = new Poodle(); // create a Poodle object
      d.size = 25; // set the size of the dog
      d.sing(); // call the sing() method of Poodle
   }
}
```

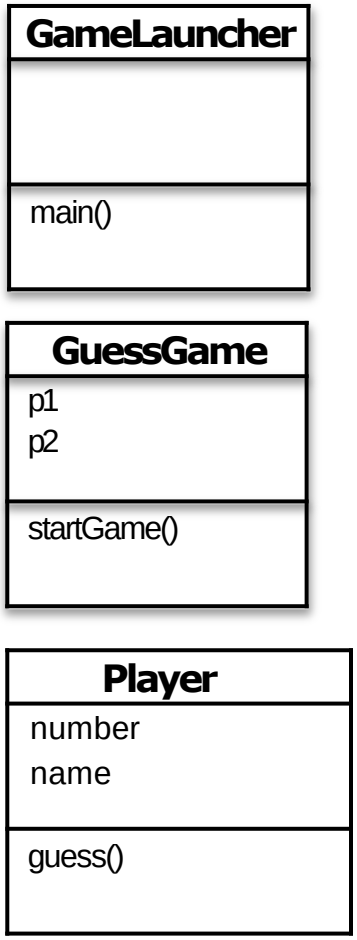
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The method sing() is undefined for the type Dog

# Example: A Guessing Game

- Summary
  - The guessing game involves
    - 1 ‘game’ object and
    - 2 ‘player’ objects
  - The game generates a random number between 0 and 9
  - The 2 players try to guess this number
  - The game ends when either or both players make a correct guess, otherwise the game continues

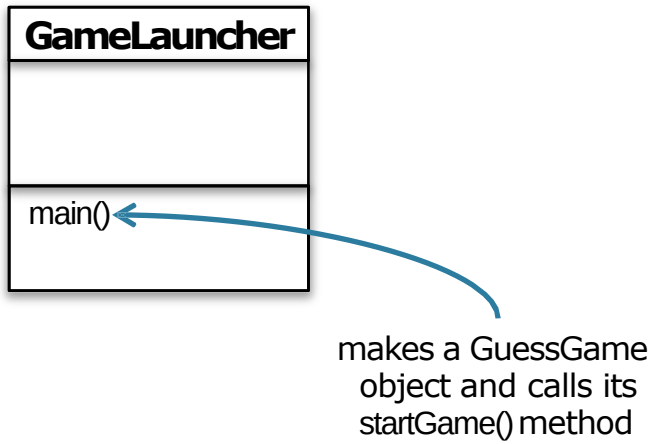
# Example: A Guessing Game

- Classes
  - GameLauncher
  - GuessGame
  - Player



# Example: A Guessing Game

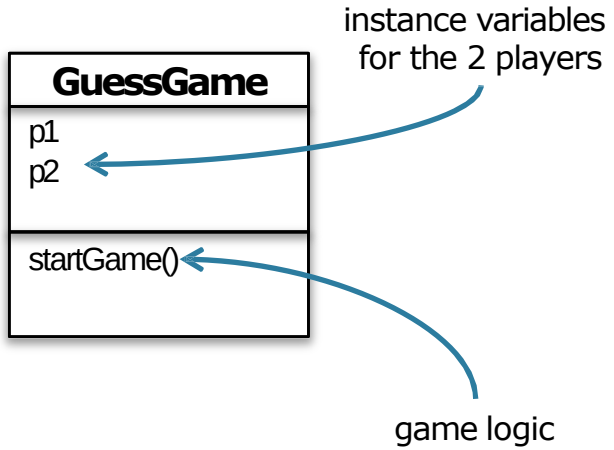
- GameLauncher class
  - Used to launch the game
  - main() method
    - Create a GuessGame object
    - Start the game by calling the startGame() method of the GuessGame object




```
public class GameLauncher {
   public static void main(String[] args) {
      GuessGame g = new GuessGame();
      g.startGame();
   }
}
```

# Example: A Guessing Game

- GuessGame class
  - Instance variables for the 2 players
  - startGame() method
    - Create 2 Player objects
    - Generate a random number
    - Ask each player to make a guess
    - Check the results
      - Print out winning message; or
      - Ask the players to make a guess again



# Example: A Guessing Game

- GuessGame class

```
public class GuessGame {
   Player p1;
   Player p2;

   public void startGame() {
      p1 = new Player();
      p1.name = "Player 1";
      p2 = new Player();
      p2.name = "Player 2";

      int targetNumber = (int) (Math.random() * 10);
      System.out.println("Target number is " + targetNumber);
```

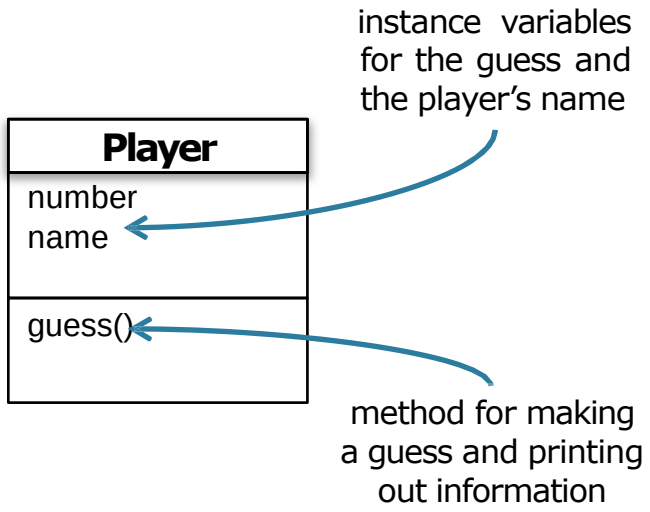
# Example: A Guessing Game

- GuessGame class

```
        boolean isFinished = false;
        while (!isFinished) {
           p1.guess();
           p2.guess();
           if (p1.number == targetNumber) {
              System.out.println(p1.name + " got it right!");
              isFinished = true;
           } // end if
           if (p2.number == targetNumber) {
              System.out.println(p2.name + " got it right!");
              isFinished = true;
           } // end if
         } // end while
      } // end startGame()
}
```

# Example: A Guessing Game

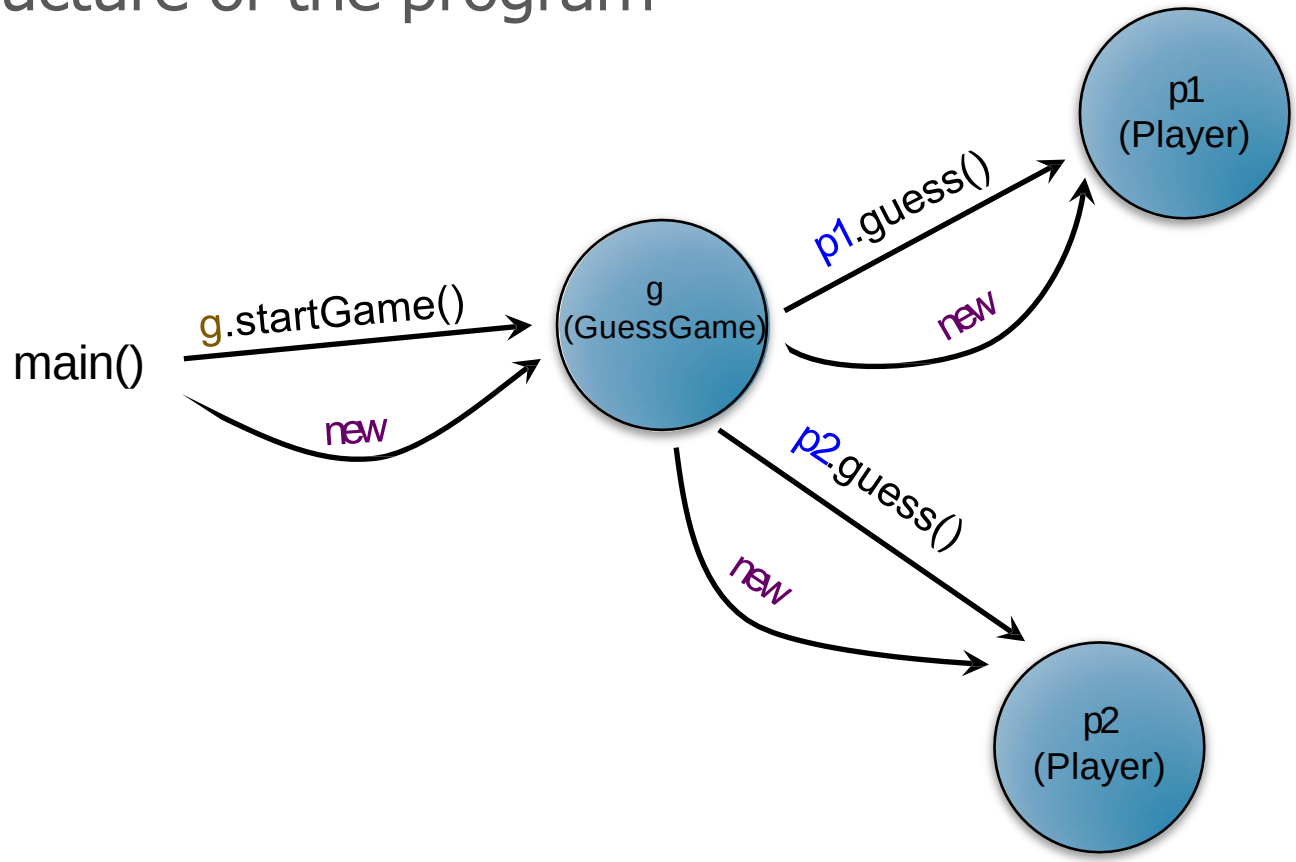
- Player class
  - Instance variables for the guess and the player's name
  - guess() method
    - Make a guess
    - Print out the player's name and his guess




```
public class Player {
   int number = 0; // where the guess goes
   String name = "Player";
   public void guess() {
      number = (int) (Math.random() * 10);
      System.out.println(name + " guessed " + number);
   }
}
```

# Example: A Guessing Game

- Structure of the program



# Example: A Guessing Game

- Sample output

```
Target number is 8
Player 1 guessed 5
Player 2 guessed 4
Player 1 guessed 0
Player 2 guessed 4
Player 1 guessed 4
Player 2 guessed 6
Player 1 guessed 6
Player 2 guessed 1
Player 1 guessed 5
Player 2 guessed 4
Player 1 guessed 8
Player 2 guessed 3
Player 1 got it right!
```

# main() Method

- A Java application is nothing but objects talking to each other
- Start running by executing the main() method of the launching class
- In most Java application, the main() method does 2 and only 2 things
  - Create an object
  - Call a method of the object

# main() Method

- Unlike C++
  - The main() method does not return any value
  - The name of the program is not included in the arguments (it is exactly the name of the class that contains the main() method!)
  - The number of arguments needs not be included (an array in Java is an object who knows its own size!)

Java:

```
public static void main(String[] args)
```

C++:

```
int main(int argc, char* argv[])
```

argument count

argument vector
(argv[0] contains the program name)

# main() Method

- A Java application can accept any number of arguments from the command line
- This allows the user to specify configuration information when an application is launched

```
public class Echo {
   public static void main(String[] args) {
      for (String arg : args) {
         System.out.println(arg);
      }
   }
}
```

# Garbage Collection

- Each time an object is created in Java, it goes into an area of memory known as the heap
- Java allocates memory space on the heap according to how much space that particular object needs (e.g., an object with 15 instance variables will probably need more space than an object with only 2 instance variables)
- Unlike C++, no need to explicitly free up memory using the delete operator
- Java manages the memory for you

# Garbage Collection


- When an object can never be used again, it becomes eligible for garbage collection
- When the garbage is actually collected can be unpredictable
- When the system is running low on memory, the Garbage Collector will run, throw away those unreachable objects to free up the memory space
- The Java heap is therefore called the Garbage-Collectible Heap

# We are with you!


If you encounter any problems in understanding the materials in the lectures, **please feel free to contact me or my TAs**.
**We are always with you!**

We wish you enjoy learning Java in this class.

# Chapter 2.


# End

**2020-2021**

**COMP2396 Object-Oriented Programming and Java**

**Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)**

**Department of Computer Science, The University of Hong Kong**