JAVA

Introduction

- Java is high-level, object-oriented programming language developed by Sun microsystem in 1991.
- Earlier name was oak given by James Gosling(father of java).
- Java use for making desktop application, web application, enterprise application, mobile app, embed system, robotics, games etc.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

Java Buzzwords

- **Simple:** similar to C and C++ also pointers, operator overloading are removed
- **Object Oriented:** can make software based on objects that support encapsulation, inheritance, polymorphism.
- **Robust:** has type checking, exception handling, lack of pointers makes java robust.
- **Secure:** absence of pointers make memory allocation difficult, access restrictions (public, private) etc.
- **Portable:** can carry bytecode to any platform and execute it.
- **Compiled and interpreted:** first java program compiled in bytecode then again interpreted by java interpreter which converts it into machine language.

Java Architecture

Java simple program

firstProgram.java

```
1  public class firstProgram {
2      public static void main(String args[]){
3          System.out.println("this is my first java program");
4      }
5  }
```

Explanation

1. Public is access modifier which tells our class firstProgram is visible to every other class
2. firstProgram is a class name.
3. Main() is our method with type void and string arguments .
4. Java main() method is always static, so that compiler can call it without the creation of an object or before the creation of an object of the class.

Arrays

- Java array is an object which contains elements of a similar data type.
- The elements of an array are stored in a contiguous memory location.
- Array has indexes from 0 to SIZE-1.
- There are two types of array:
  1. Single-dimensional array
  2. Multi dimensional array

Single-dimensional Array

- One-dimensional array in Java programming is an array with a bunch of values having been declared with a single index.

```
import java.util.Scanner;

public class arraycode {
   public static void main(String args[]) {
      System.out.println("[+++++++++++++++method 1+++++++++++++++++]");
      int myarr[] = {1, 2, 3, 4, 5};
      System.out.println(myarr[2]);


      System.out.println("[+++++++++++++++method 2+++++++++++++++++]");
      int myarr2[] = new int[5];
      myarr2[0] = 1;
      myarr2[1] = 2;
      myarr2[2] = 3;
      myarr2[3] = 4;
      myarr2[4] = 5;
      System.out.println(myarr[2]);


      System.out.println("[+++++++++++++++method 3+++++++++++++++++]");
      int myarr3[] = new int[5];
      System.out.println("Enter the array elements");
      Scanner sc = new Scanner(System.in);
      for (int i = 0; i < 5; i++) {
         myarr3[i] = sc.nextInt();
      }
      System.out.println("the array elements are");
      for (int i = 0; i < 5; i++) {
         System.out.println(myarr3[i]);
      }

   }
}
```

Multi-dimensional Array

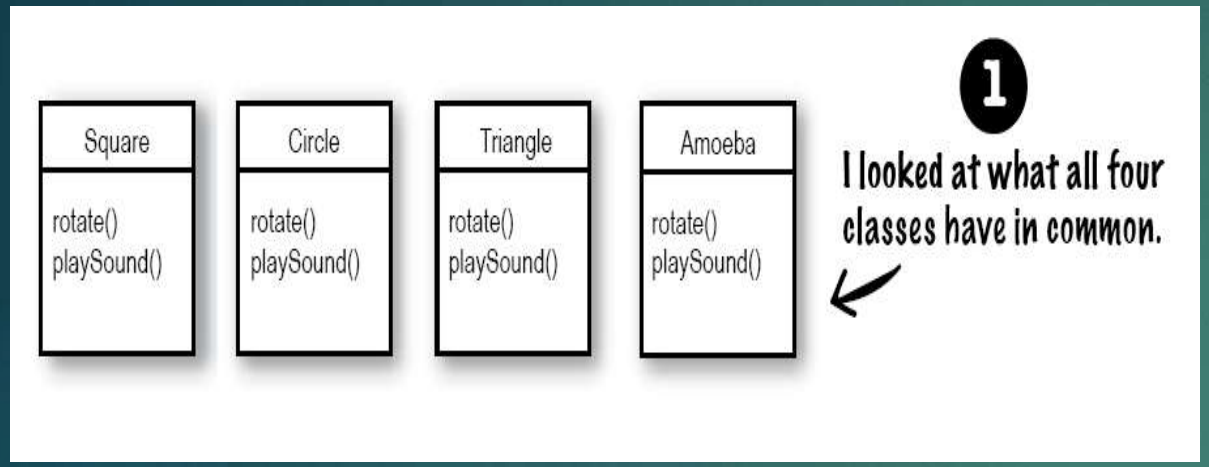
- One-dimensional array in Java programming is an array with a bunch of values having been declared with a multiple index.

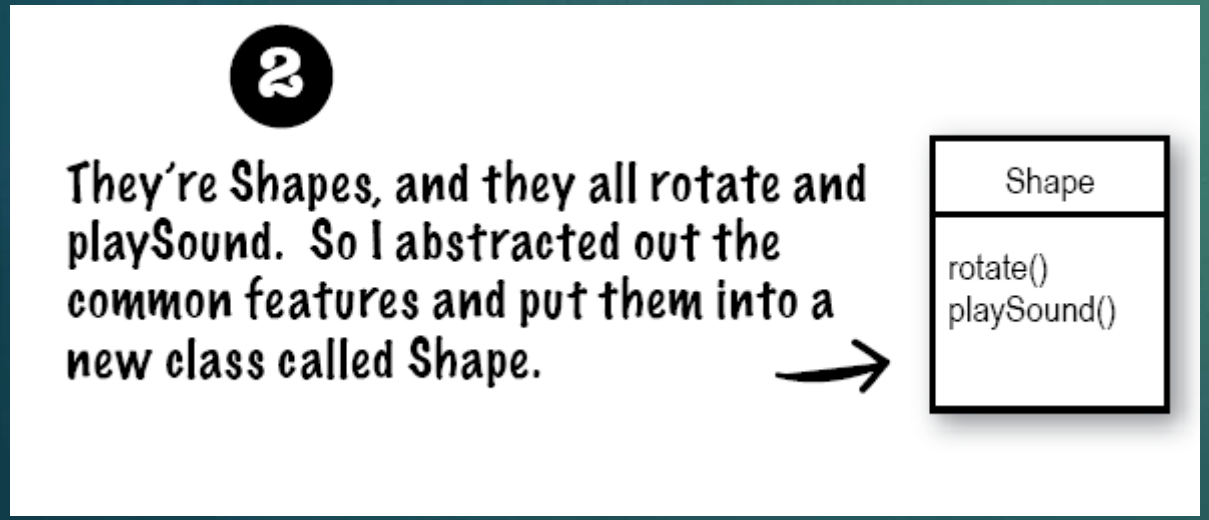
```
import java.util.Scanner;

public class arraycode {
   public static void main(String args[]) {
      System.out.println("[+++++++++++++++method 1+++++++++++++++++]");
      int myarr[] = {1, 2, 3, 4, 5};
      System.out.println(myarr[2]);


      System.out.println("[+++++++++++++++method 2+++++++++++++++++]");
      int myarr2[] = new int[5];
      myarr2[0] = 1;
      myarr2[1] = 2;
      myarr2[2] = 3;
      myarr2[3] = 4;
      myarr2[4] = 5;
      System.out.println(myarr[2]);


      System.out.println("[+++++++++++++++method 3+++++++++++++++++]");
      int myarr3[] = new int[5];
      System.out.println("Enter the array elements");
      Scanner sc = new Scanner(System.in);
      for (int i = 0; i < 5; i++) {
         myarr3[i] = sc.nextInt();
      }
      System.out.println("the array elements are");
      for (int i = 0; i < 5; i++) {
         System.out.println(myarr3[i]);
      }

   }
}
```

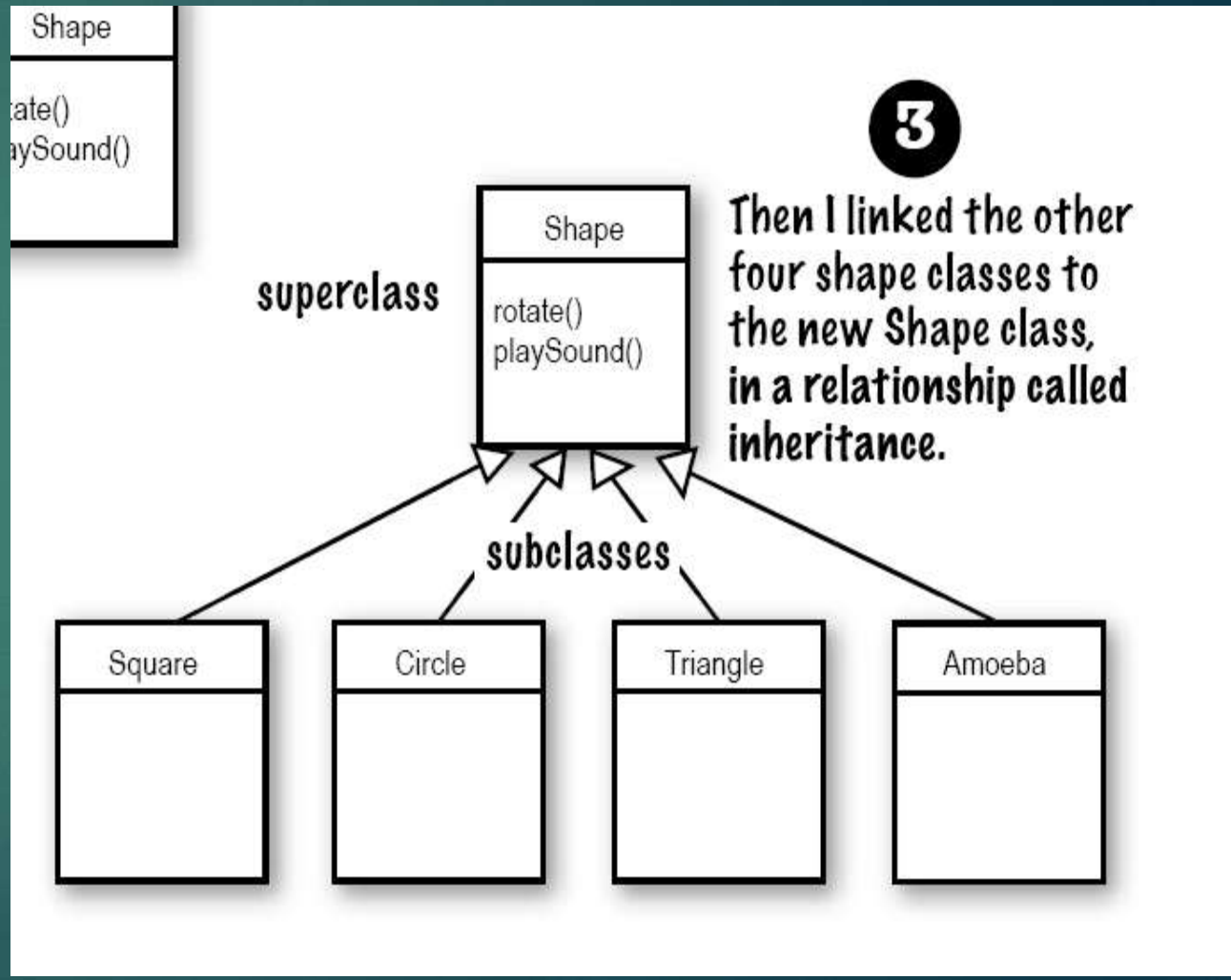
Inheritance

- Inheritance is one of the core concepts of Object-Oriented Programming.
- It allows a class to use the properties and methods of another class.
- The purpose of inheritance in java, is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be inherited from the another class.

Understanding inheritance

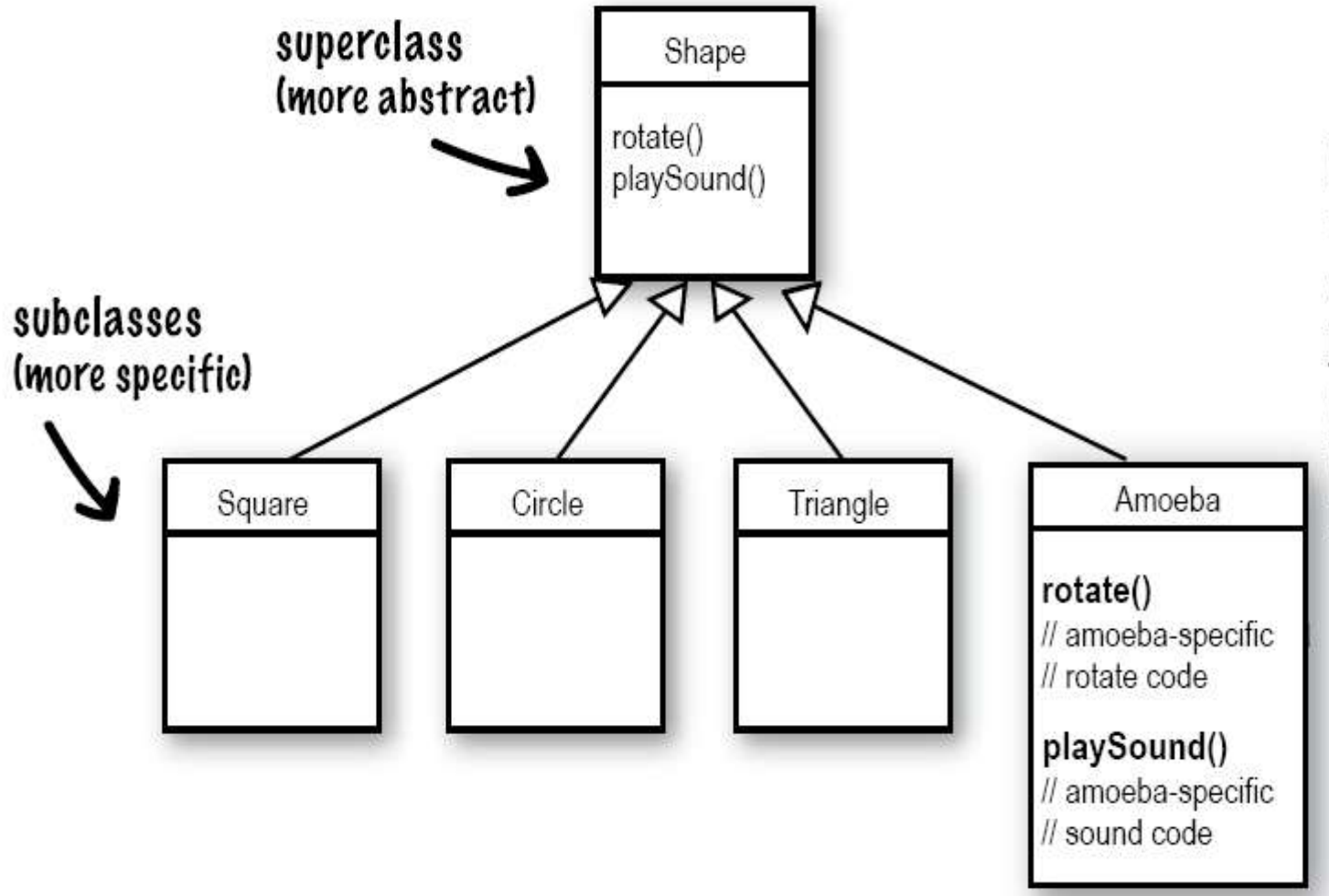






- Square, Circle, Triangle and Amoeba inherits from Shape class

The subclass can override Super class method



4

I made the Amoeba class override the rotate() and playSound() methods of the superclass Shape. Overriding just means that a subclass redefines one of its inherited methods when it needs to change or extend the behavior of that method.

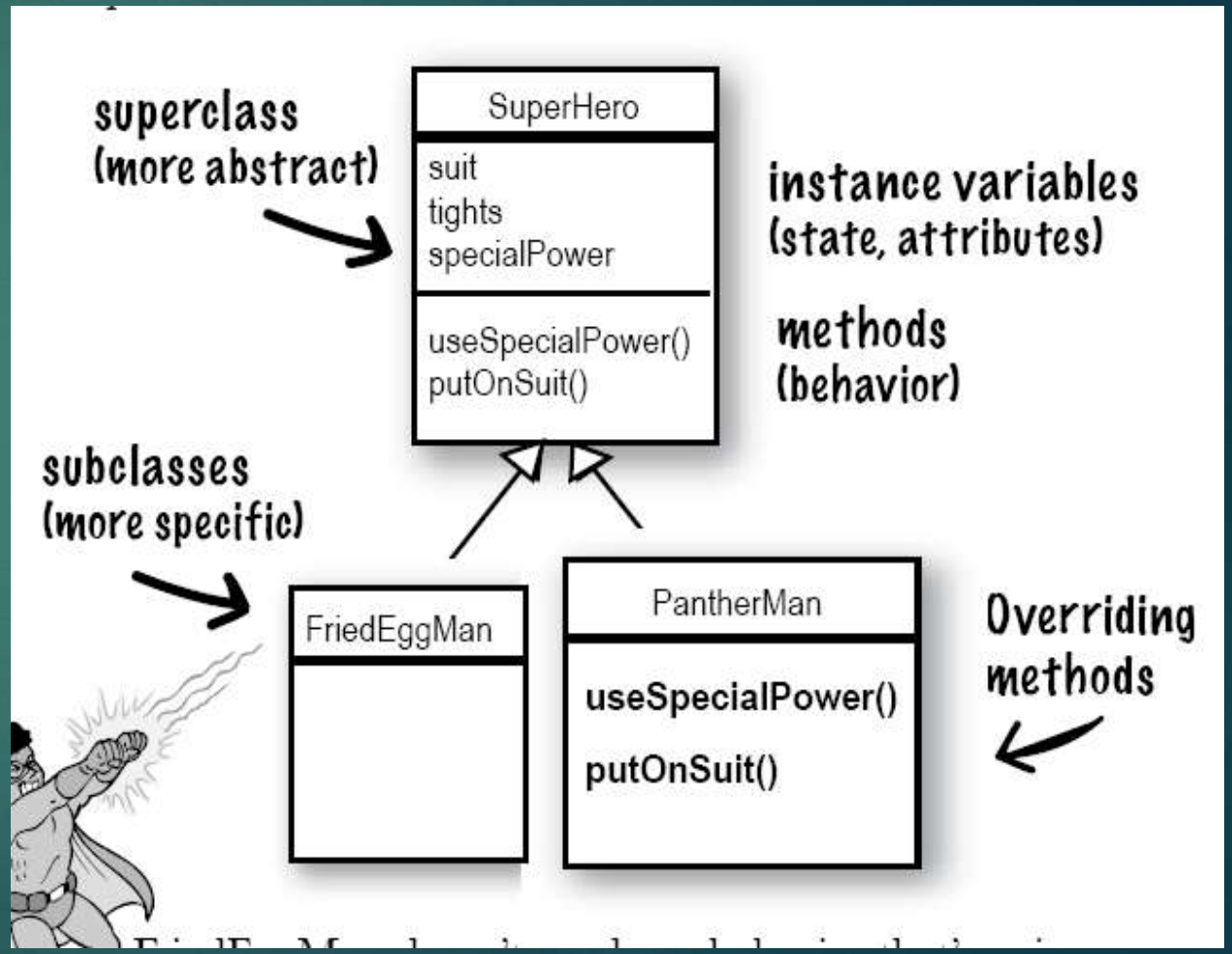
Overriding methods

Example 2

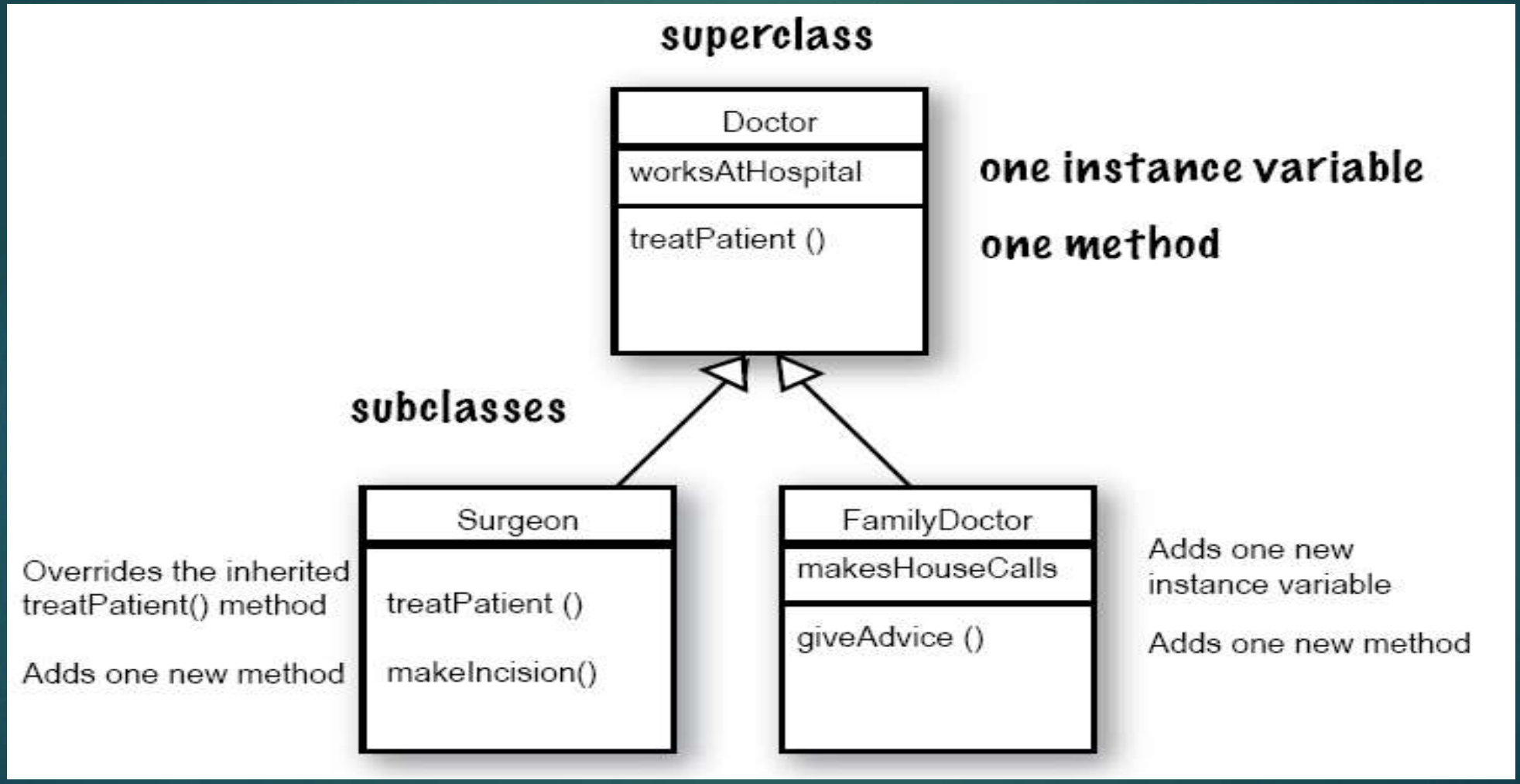
- Some class just inherit from superclass and some class can override the methods.
- Instance variable are not

overridden because they not need

to, any subclass can give it's value .

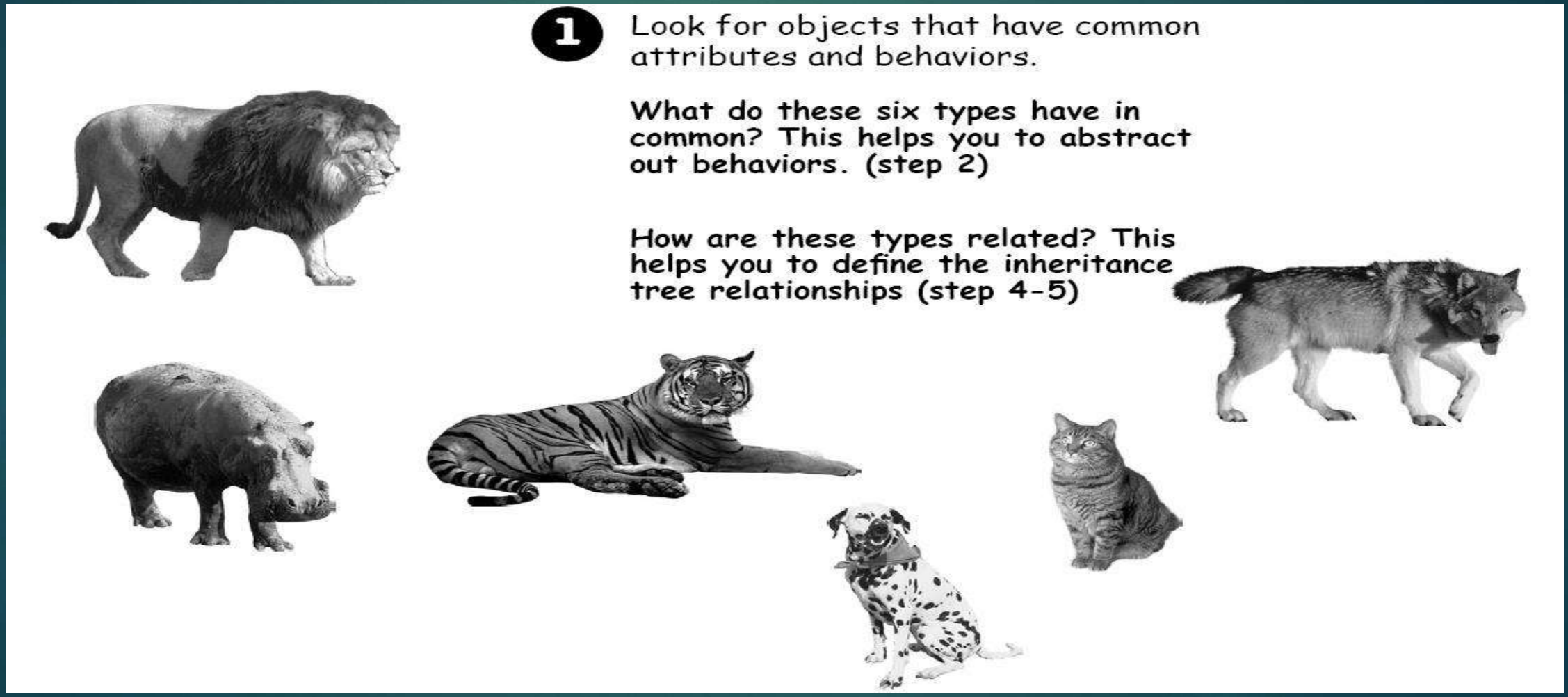


Example 3

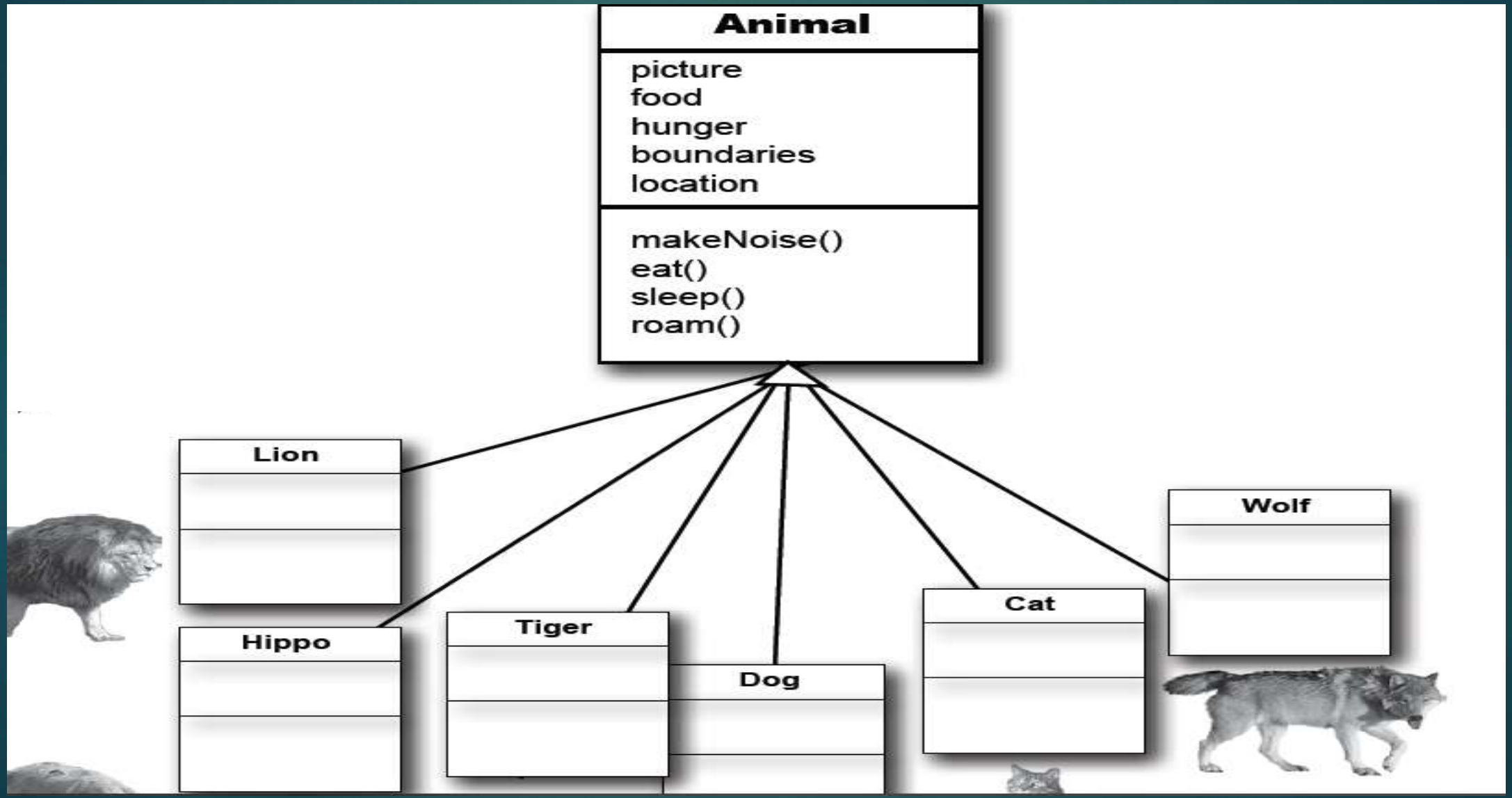

superclass
Doctor
worksAtHospital
treatPatient ()
one instance variable
one method
subclasses
Surgeon
treatPatient ()
makeIncision()
Overrides the inherited treatPatient() method
Adds one new method
FamilyDoctor
makesHouseCalls
giveAdvice ()
Adds one new instance variable
Adds one new method

Animal Simulation

- Inheritance designing for animal

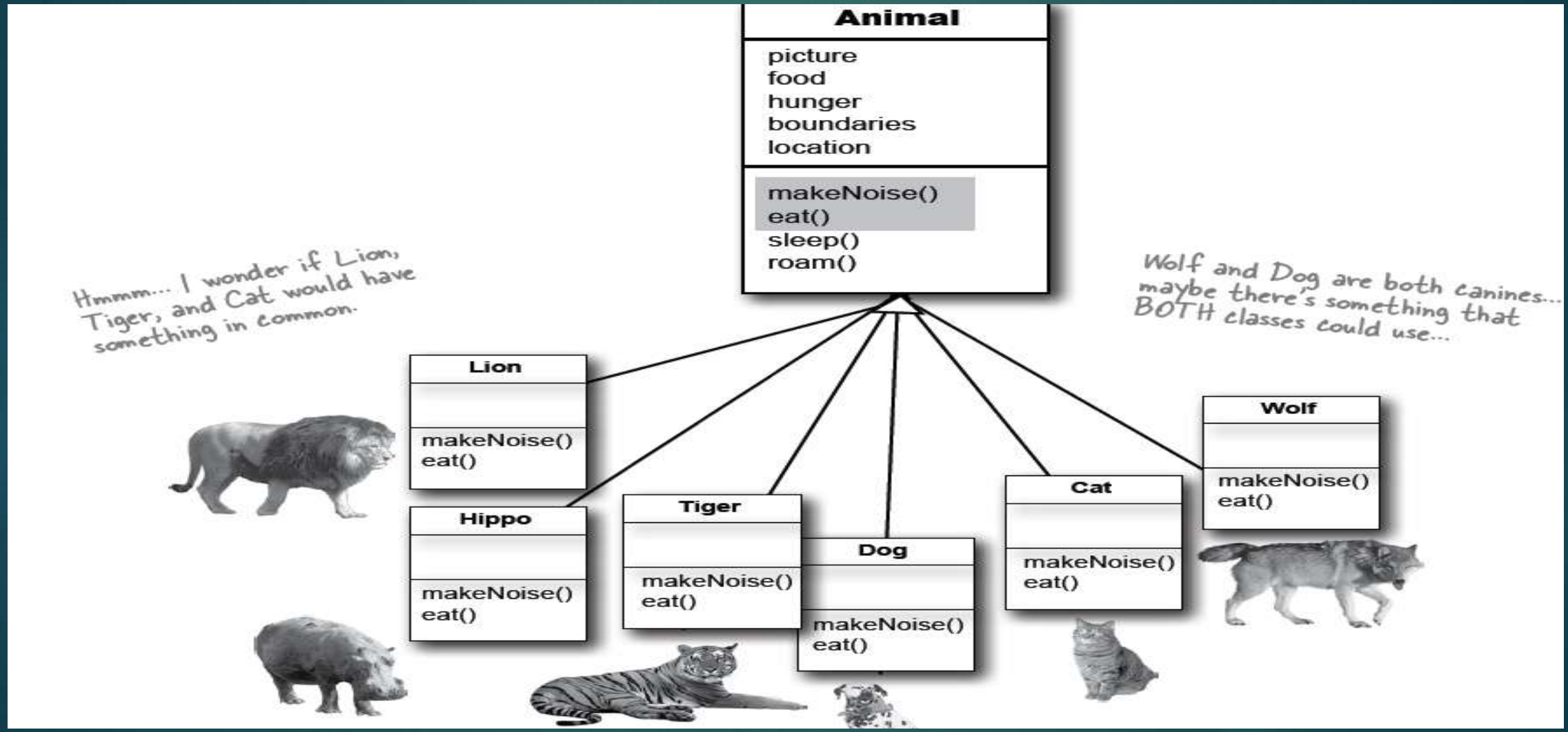


Animal Simulation 2



Animal Simulation 3

- Overriding makeNoise() and eat() methods because all animal have different property



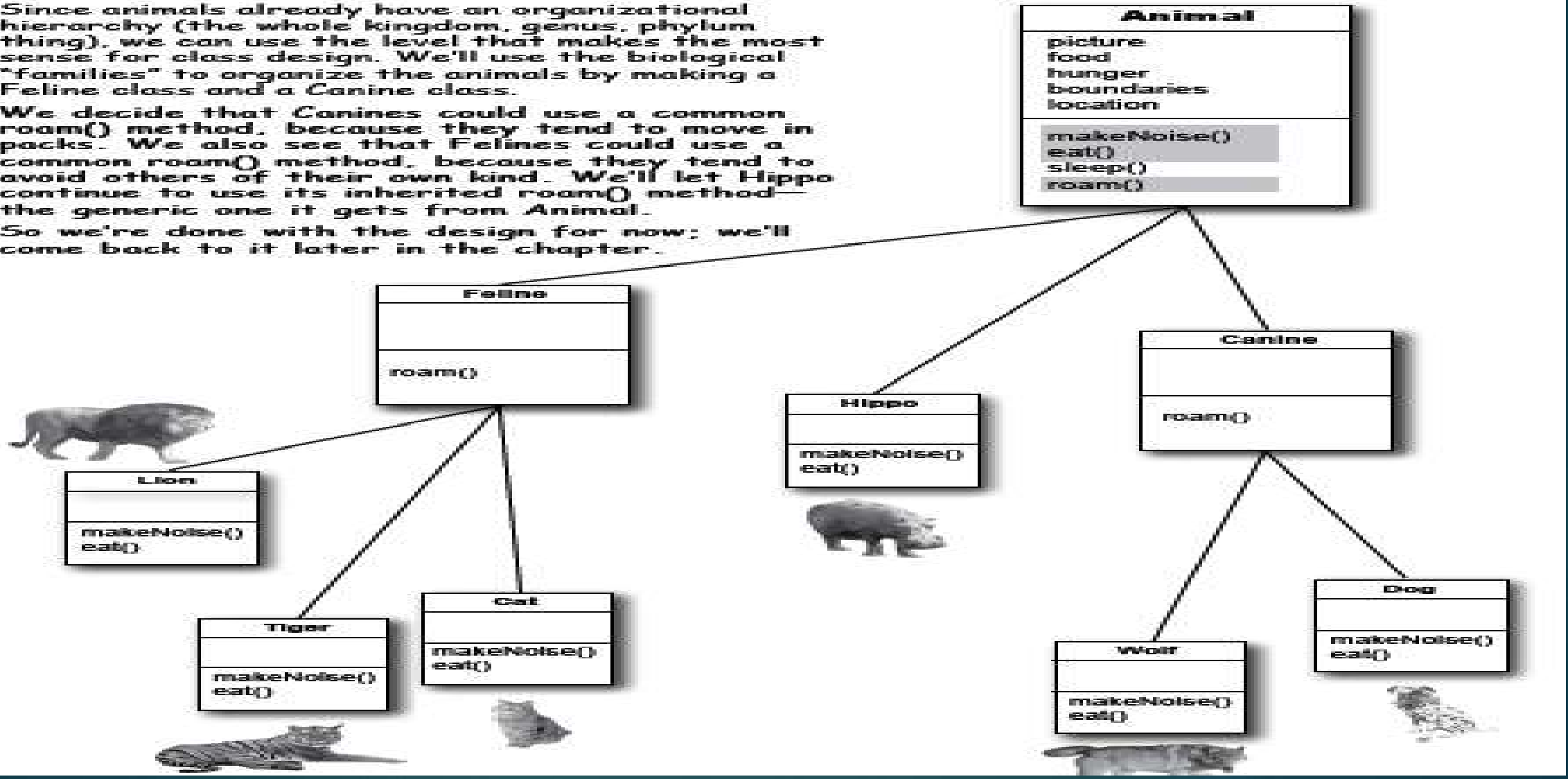
Animal Simulation 3

Further dividing lion, tiger and cat in one feline group and wolf, dog in canine

Since animals already have an organizational hierarchy (the whole kingdom, genus, phylum thing), we can use the level that makes the most sense for class design. We'll use the biological "families" to organize the animals by making a Feline class and a Canine class.

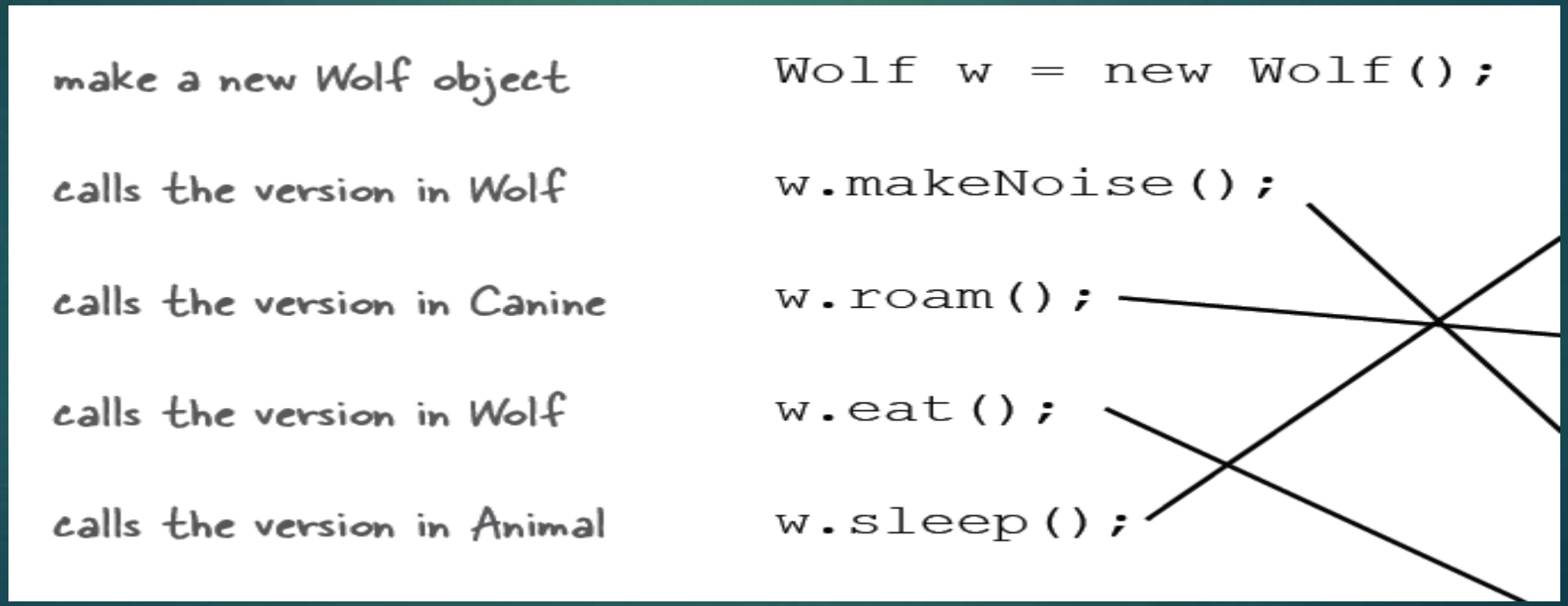
We decide that Canines could use a common roam() method, because they tend to move in packs. We also see that Felines could use a common roam() method, because they tend to avoid others of their own kind. We'll let Hippo continue to use its inherited roam() method—the generic one it gets from Animal.

So we're done with the design for now; we'll come back to it later in the chapter.



Which methods are called?

- JVM takes cares which method to execute on run time.
- The lowest specific one executed first and reverse one step backward.



Using Is-A and Has-A

- Consider this,
  - is Cat an Animal ? Yes-> Cat is subclass and Animal is superclass

    -> or we can say Cat extends Animal
  - is Animal a Cat ? No
  - is room a house ? No
  - is house a room ? No
  - House has-a room? Yes -> so house is a member variable of class Room

  - Conclusion: **is-a** is use to test superclass and subclass relationship whereas **has-a** Is use to check members variable.

If subclass wants both superclass and its method code to override

superclass version using the keyword **super**. It's like saying, "first go run the superclass version, then come back and finish with my own code..."

A: You can do this! And it's an important design feature. Think of the word "extends" as meaning, "I want to *extend* the functionality of the superclass".

```
public void roam() {
    super.roam();
     // my own roam stuff
}
```

this calls the inherited version of roam(), then comes back to do your own subclass-specific code

There are 4 access-control

- Private
- Default
- Protected
- Public
  - For now we deal with private and public only
  - Private makes outside none can use it and public means anyone can use it.

BULLET POINTS

- A subclass *extends* a superclass.
- A subclass inherits all *public* instance variables and methods of the superclass, but does not inherit the *private* instance variables and methods of the superclass.
- Inherited methods *can* be overridden; instance variables *cannot* be overridden (although they can be *redefined* in the subclass, but that's not the same thing, and there's almost never a need to do it.)
- Use the IS-A test to verify that your inheritance hierarchy is valid. If X *extends* Y, then X *IS-A* Y must make sense.
- The IS-A relationship works in only one direction. A Hippo is an Animal, but not all Animals are Hippos.
- When a method is overridden in a subclass, and that method is invoked on an instance of the subclass, the overridden version of the method is called. (*The lowest one wins.*)
- If class B extends A, and C extends B, class B IS-A class A, and class C IS-A class B, and class C also IS-A class A.

Advantages of inheritance

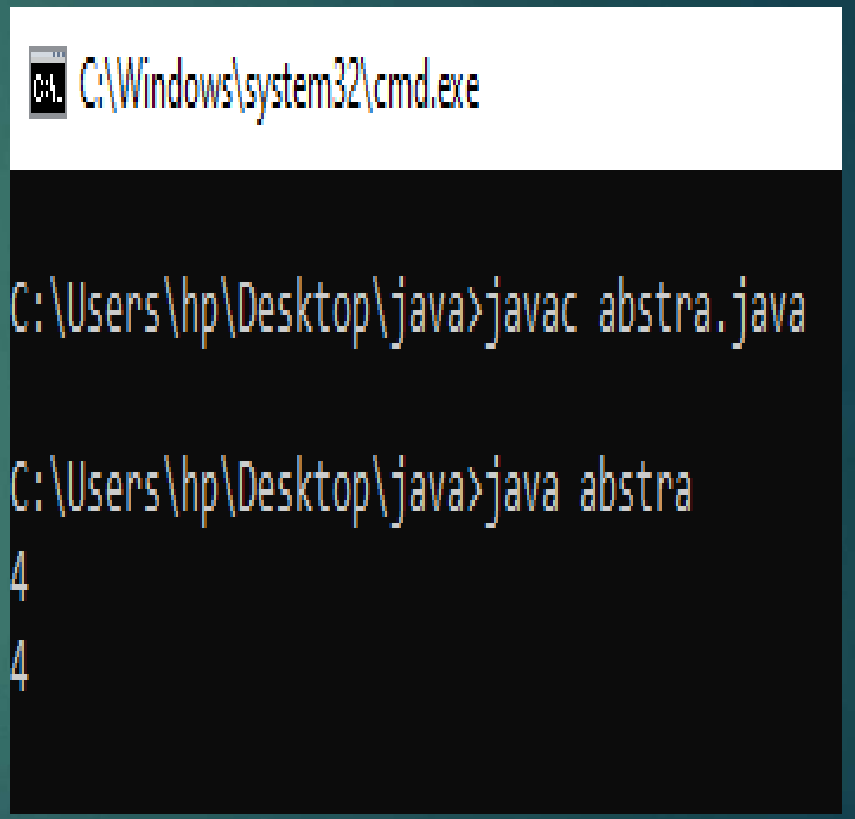
- Avoiding duplicate code
- Defining a common protocol for a group of classes.

Abstract class and methods

- Abstract class prevents the class from being instantiated.
- When creating object of class doesn't make sense, we can make that class abstract.
- Eg. Consider a class Animal and we creating an object of that class animal, which doesn't make any sense because we don't know what kind of animal is that.
- So we make that class animal abstract , compiler prevents from animal class being instantiated.
- So we must create subclass that extends this class to create the object and use.

Abstract class and methods

```
abstract class Shape2{...}

3 usages
class Rectangle2 extends Shape2{

}
public class abstra {
    public static void main(String args[]){
        Shape2 s2 = new Rectangle2();
        s2.setData( l: 2, b: 2, h: 2);
        System.out.println(s2.findArea());
        Rectangle2 r2 = new Rectangle2();
        r2.setData( l: 2, b: 2, h: 2);
        System.out.println(r2.findArea());
    }
}
```



Abstract class and methods

- Abstract method means method must be overridden.
- It contains declaration only, no implementation.

  Public abstract findArea();

- If class contains abstract method, class must be abstract too.
- Keeping abstract method doesn't make sense but use to make protocol for subclass.
- Abstract method must be implement in subclass

- It’s a way to prevent a closs from every being instaniated.
- Compiler stop the code on creating new.

interface

- An **interface in Java** is a blueprint of a class.
- It has static constants and abstract methods.
- It is a reference type similar to class but contains only constants and methods declaration.

Interface coding

interfaceDemo.java | interfaces.java | multipleInheri.java

```
interface Animal{
    2 usages
    public void playNoise();
}

class Dog implements Animal{
    2 usages
    public void playNoise() { System.out.println("bark bhau bhau"); }
}

class interfaces{
    public static void main(String args[]){
        Dog d1 = new Dog();
        d1.playNoise();
    }
}
```

```
interface Animal {
   void playNoise();

   void knowFood();
}

interface AnimalLocation{
   void getLocation();
}

class Dog implements Animal, AnimalLocation {
   private String name;
   private String loc;
   Dog() {
   }
   Dog(String name, String l) {
      this.name = name;
      this.loc = l;
   }
   public void playNoise() {
      System.out.println("bark bhau bhau");
   }
   public void knowFood() {
      System.out.println("meat\t\trice\t\tbread");
   }
   public void getName() {
      System.out.println("name is :  " + name);
   }
   public void getLocation(){
      if (this.loc=="jungle")
         System.out.println("animal found in jungle");
      else
         System.out.println("animal found in home");
   }
}

class interfaceDemo {
   public static void main(String args[]) {
      Dog d1 = new Dog("jacky", "home");
      d1.playNoise();
      d1.knowFood();
      d1.getName();
      Dog d2 = new Dog();
      d2.getName();
      d1.getLocation();
   }
}
```

Multiple inheritance

```
//implementing muliple inheritance
interface Father {
   void knowJob();

   void knowName();
}
interface Mother extends Father{
   void knowJobM();
}

class College{
   void getCollege(){
      System.out.println("coolege method");
   }
}
class Child extends College implements  Mother{
   public void knowJob(){
      System.out.println("know job");
   }
   public void knowJobM(){
      System.out.println("know job mother");
   }
   public void knowName(){
      System.out.println("know father name");
   }
}

public class multipleInheri {
   public static void main(String args[]){
      Child c1 = new Child();
      c1.knowJob();
      c1.knowJobM();
      c1.knowName();
      c1.getCollege();
   }
}
```

Packages

- Every class in the java belongs to the packages
- Eg: ArrayList is in package called java.util, System(System.out.println), String, Math (Math.random()) all this belongs to java.lang package.

Importance of package

1. It help to organize a project or library, rather than having horrendously large pile of classes, they are grouped into packages.
2. It gives name scoping, to help prevent collisions if different programmers trying to give same name to the class they have to tell the JVM which class they are using.
3. It provide a level of security so that the class in the package can only access the code.

How to use package?

1. IMPORT

   Put the import statement at the top of the code.

   Import java.util.ArrayList;

2. TYPE

   type the full name in the code anywhere you use it.

   java.util.ArrayList<Dog> list = new java.util.ArrayList<Dog>();

Package Demo

implementation with and without package

packDemo.java

```java
import java.lang.String;
import java.lang.System;

public class packDemo {
    public static void main(String[] args) {
        java.lang.String name = "using full package name";
        String name1 = "no need to use full package name";
        System.out.println(Math.random() * 10);
        java.lang.System.out.println(name);
        System.out.println(name1);

        java.util.Scanner sc = new java.util.Scanner(System.in);
        int x = sc.nextInt();
        System.out.println(x);
        Scanner sc1 = new Scanner(System.in);
        int y = sc1.nextInt();
        System.out.println(y);
    }
}
```

```
C:\Users\hp\Desktop\java\program>javac packDemo.java
packDemo.java:15: error: cannot find symbol
        Scanner sc1 = new Scanner(System.in);
        ^
  symbol:   class Scanner
  location: class packDemo
packDemo.java:15: error: cannot find symbol
        Scanner sc1 = new Scanner(System.in);
                          ^
  symbol:   class Scanner
  location: class packDemo
2 errors
```

Keep class in package

- Select package name in reverse hite.digital
- Create folder classes in root folder. i.e classes and src in same folder
- Put source code .java in src/hite.digital folder/package
- Write package hite.digital; at 1st line in every .java file.
- Cd src
- Javac –d ../classes hite/digita/shape.java
- Or
- javac -d ../classes hite/digital/*.java
- Cd ..
- Cd classes
- Java hite.digital.shape
- Other class can import it and use the code

Shape.java

```java
package hite.digital;
import java.util.Scanner;
5 usages
public class Shape {
    1 usage
    public void sum(){
        int a, b;
        Scanner sc= new Scanner(System.in);
        System.out.println("enter any 2 numbers");
        a = sc.nextInt();
        b = sc.nextInt();
        System.out.println("sum = " + (a+b));
    }

    public static void main(String[] args) {...}
}
```

Arary.java | inheritanceUse.java | inheritanceUse.class

```java
package hite.digital;
import hite.digital.Shape;
import java.util.Scanner;

2 usages
public class Arary {
    1 usage
    public void sorting(){...}

    public static void main(String[] args)
        System.out.println("aray main clas
        Shape s1 = new Shape();
        s1.sum();
    }
}
```

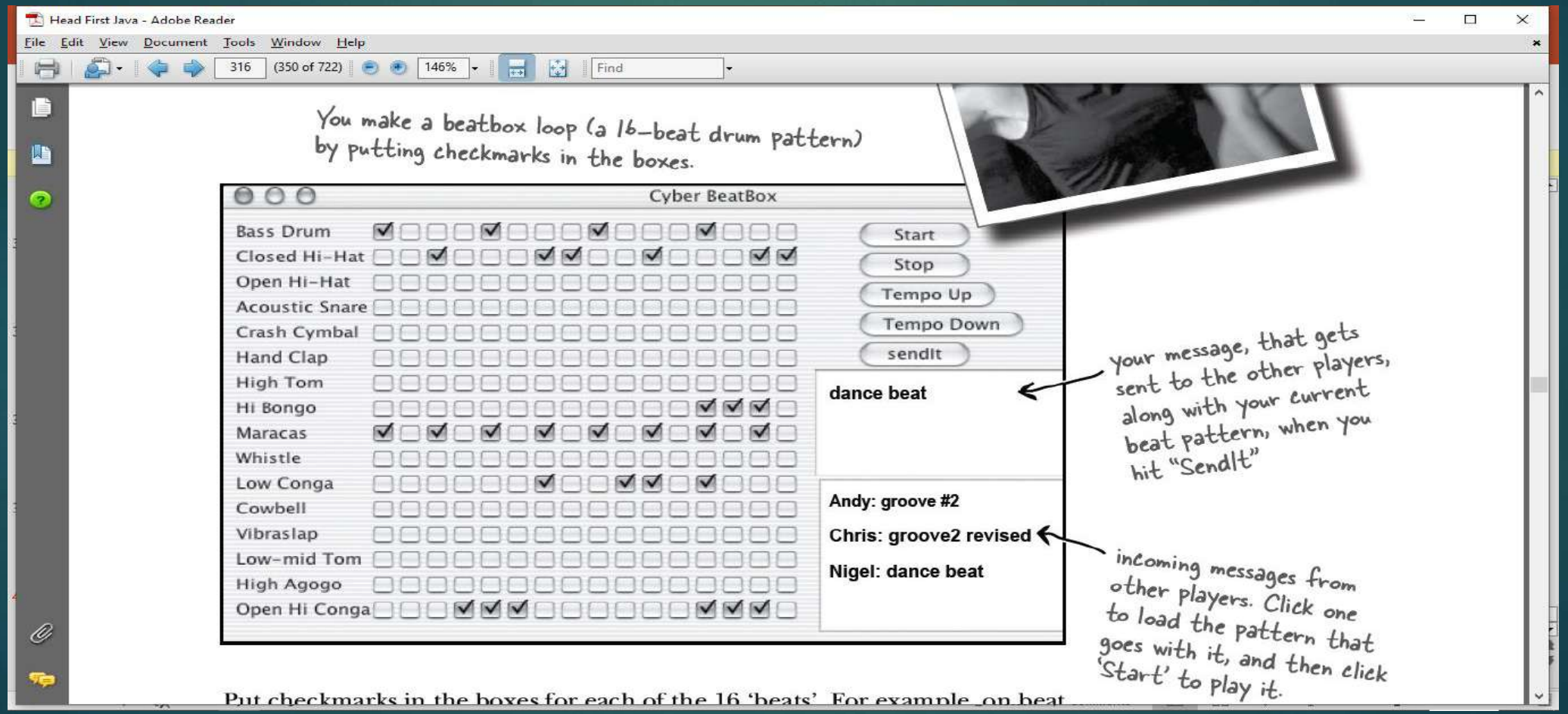
C:\Windows\system32\cmd.exe

```
C:\Users\hp>cd C:\Users\hp\Desktop\java\packagage\PackFolder\src

C:\Users\hp\Desktop\java\packagage\PackFolder\src>javac -d ../classes hite/digital/*.java

C:\Users\hp\Desktop\java\packagage\PackFolder\src>cd..

C:\Users\hp\Desktop\java\packagage\PackFolder>cd classes

C:\Users\hp\Desktop\java\packagage\PackFolder\classes>java hite.digital.Arary
aray main class
enter any 2 numbers
2
3
sum = 5
```

EXCEPTION HANDLING

- there might be problem in the run time, so programmers must handle the risky method.
- Example file not found, server down, divide by zero etc.

Learning exception handling with making music machine



Features include music app

- It include GUI Swing,
- How to connect another machine
- How to use i/o for sending message to another machine

Javasound API

- JavaSound is a collection of classes and interfaces.
- JavaSound is split into 2 parts MIDI and sampled.
- MIDI stands for Musical Instrument Digital Interface.
- MIDI is like a sheet of music, it doesn't include sound, it contains instructions only.
- MIDI file is like HTML document and device that renders it is like web browser.
- Instrument may be built by software like digital keyboard.
- We use Synthesizer software that creates sound.

Need Sequencer

- We need a sequencer object
- Its an object that takes all the midi data and sends it to the right instruments.

```java
import javax.sound.midi.*;

2 usages
public class Main {
    1 usage
    public void play() {
            Sequencer sequencer = MidiSystem.getSequencer();
            System.out.println("we got a sequencer");
            System.out.println("hummmmmmmmmmmm");
    }

    public static void main(String[] args) {
        Main m1 = new Main();
        m1.play();
    }
}
```

```
C:\Windows\system32\cmd.exe

C:\Users\hp\Desktop\java\excetional\MusicTest1\src>javac Main.java
Main.java:5: error: unreported exception MidiUnavailableException; must be caught or declared to be thrown
           Sequencer sequencer = MidiSystem.getSequencer();
                                                        ^
1 error

C:\Users\hp\Desktop\java\excetional\MusicTest1\src>
```

- Use try/catch to handle the exception code.
- Put risky code try block and catch the code if exception occur.

Try/catch

- Write risky code in try block and when that code throws exception it should put in catch block.

```
public class exp {
    public static void main(String[] args) {

        try {
            int ar[] = new int[3];
            ar[4] = 6;
        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println("array has size limit 3");
            System.out.println(ex.getMessage());
        }
    }
}
```

Multiple catch block

- Method might throw multiple exception so we should use multiple catch block.

```java
public class exp {
    public static void main(String[] args) {
        try {
            int ar[] = new int[3];
            ar[4] = 6/0;
        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println(ex.getMessage());
        }catch (ArithmeticException ex1) {
            System.out.println(ex1.getMessage());
        }
    }
}
```

```java
public class exp {
    public static void main(String[] args) {
        try {
            int ar[] = new int[3];
            ar[4] = 4 / 2;
        } catch (ArithmeticException | ArrayIndexOutOfBoundsException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Try/catch/Finally block

- We can use final block to write code that must run no matter try or catch execute.
- Even if there is return in try/catch, finally code is execute.

```java
public class exp {
    public static void main(String[] args) {
        try {
            int ar[] = new int[3];
            ar[4] = 4 / 2;
        } catch (ArithmeticException | ArrayIndexOutOfBoundsException ex) {
            System.out.println(ex.getMessage());
        } finally {
            System.out.println("i will exceute no matter what");
        }
    }
}
```

```
C:\Users\hp\Desktop\java\excetional\MusicTest1\src>javac exp.java

C:\Users\hp\Desktop\java\excetional\MusicTest1\src>java exp
Index 4 out of bounds for length 3
i will exceute no matter what
```

Throws and throw

- Exception can be 2 types:
  - Checked and unchecked:
  - Checked are check at compile time like IOException, InterruptException

  And unchecked are ignore at compile time but check at runtime like ArithmeticException, ArrayOutOfIndex.
  - Throws and throw is use to use to handle check exception.

Throws keyword

```java
import java.io.*;

public class throwws {
    1 usage
    public static void findFile() throws IOException {
        // code that may produce IOException
        File newFile = new File( pathname: "test.txt");
        FileInputStream stream = new FileInputStream(newFile);
    }

    public static void main(String[] args) {
        try {
            findFile();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

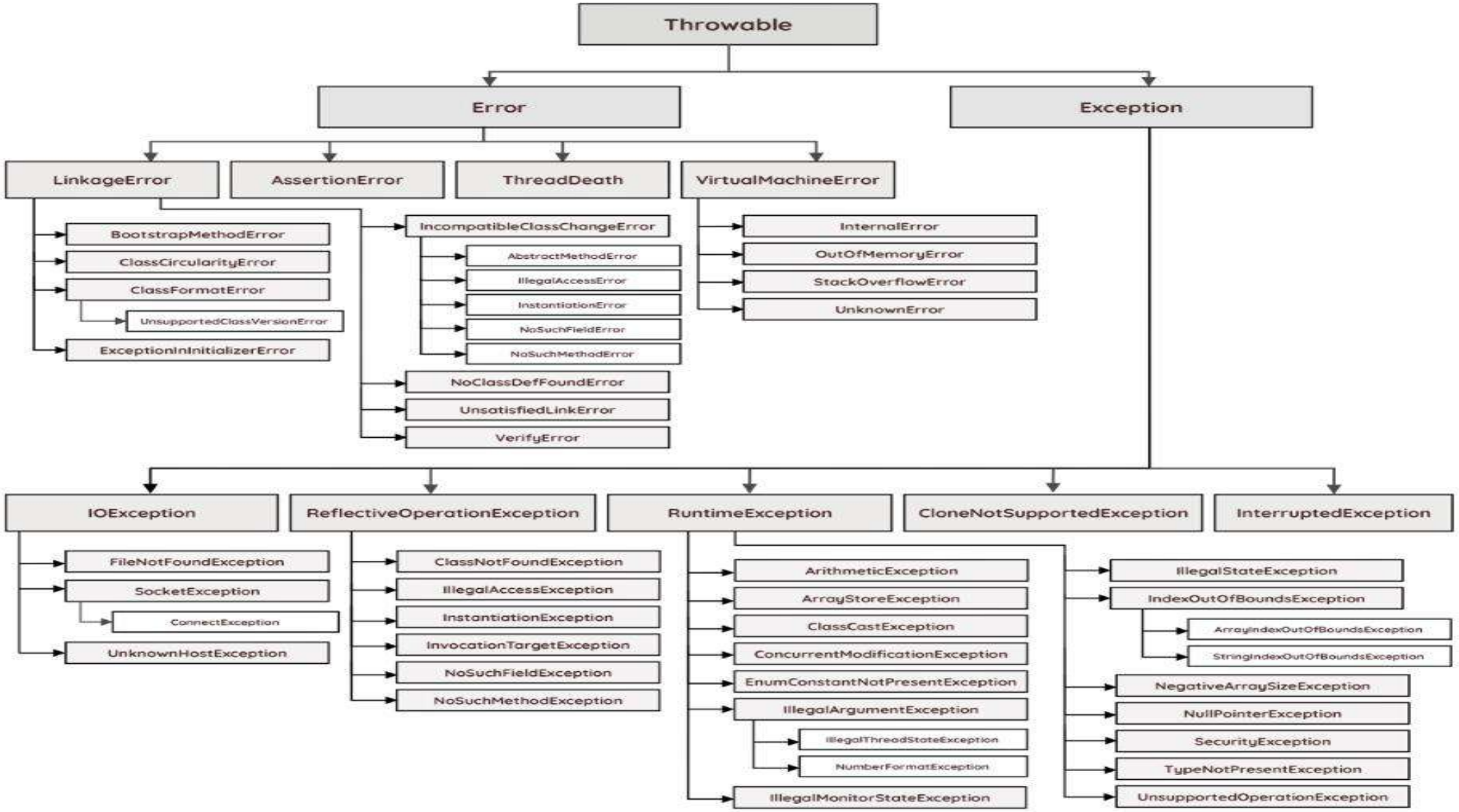
- It is use in method declaration

To declare type of exception.

Throw keyword

- is used to explicitly throw a single exception

```
public class throwws {
    1 usage
    public static void divideByZero() {
        throw new ArithmeticException("Trying to divide by 0");
    }

    public static void main(String[] args) {
        divideByZero();
    }
```


Throwable
Error
Exception
LinkageError
AssertionError
ThreadDeath
VirtualMachineError
BootstrapMethodError
ClassCircularityError
ClassFormatError
UnsupportedClassVersionError
ExceptionInInitializerError
IncompatibleClassChangeError
AbstractMethodError
IllegalAccessError
InstantiationError
NoSuchFieldError
NoSuchMethodError
NoClassDefFoundError
UnsatisfiedLinkError
VerifyError
InternalError
OutOfMemoryError
StackOverflowError
UnknownError
IOException
ReflectiveOperationException
RuntimeException
CloneNotSupportedException
InterruptedException
FileNotFoundException
SocketException
ConnectException
UnknownHostException
ClassNotFoundException
IllegalAccessException
InstantiationException
InvocationTargetException
NoSuchFieldException
NoSuchMethodException
ArithmeticException
ArrayStoreException
ClassCastException
ConcurrentModificationException
EnumConstantNotPresentException
IllegalArgumentException
IllegalThreadStateException
NumberFormatException
IllegalMonitorStateException
IllegalStateException
IndexOutOfBoundsException
ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsException
NegativeArraySizeException
NullPointerException
SecurityException
TypeNotPresentException
UnsupportedOperationException