Mahidol University
Faculty of Information
and Communication Technology

LECTURE 06

Inheritance

ITCS123 Object Oriented Programming

Dr. Siripen Pongpaichet
Dr. Petch Sajjacholapunt
Asst. Prof. Dr. Ananta Srisuphab

Semester 2/2023 15 Feb 2024

(Some materials in the lecture are done by Aj. Suppawong Tuarob)

Recap – Lecture 05

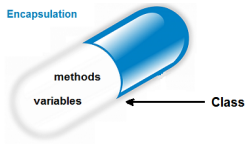
In the past week, how many **hours** you spent on coding and reviewing lecture outside the classroom?

- Solving problem using OOP
  - Discover **Classes** with case study vending machine
  - Design good **Methods**
    - Cohesive, minimize dependency, separate accessors and mutator, and minimize side effect
  - Understand **Parameters**
    - Explicit vs Implicit, OOP always pass by value but for the object, its value is the address.
  - Common patters for **Object's data** (attributes/instance variables)
    - Keeping total, collecting values, managing properties, and tracking object's state

Class Learning Outcome

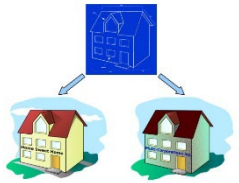
- To **explain** about inheritance
- To **implement** subclasses that inherit and overrride superclass mtehods

Major Principles of OOP



- **Encapsulation**
  - We already learned to hide unnecessary details in our classes and provide a clear and simple interface for working with them.

- **Inheritance**
  - **We will explain how class hierarchies improve code readability and enable the reuse of functionality.**

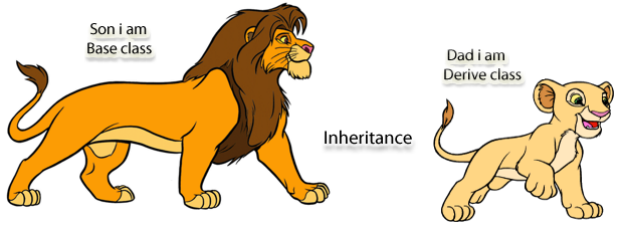


- **Abstraction**
  - We will learn how to work through abstractions: to deal with objects considering their important characteristics and ignore all other details.

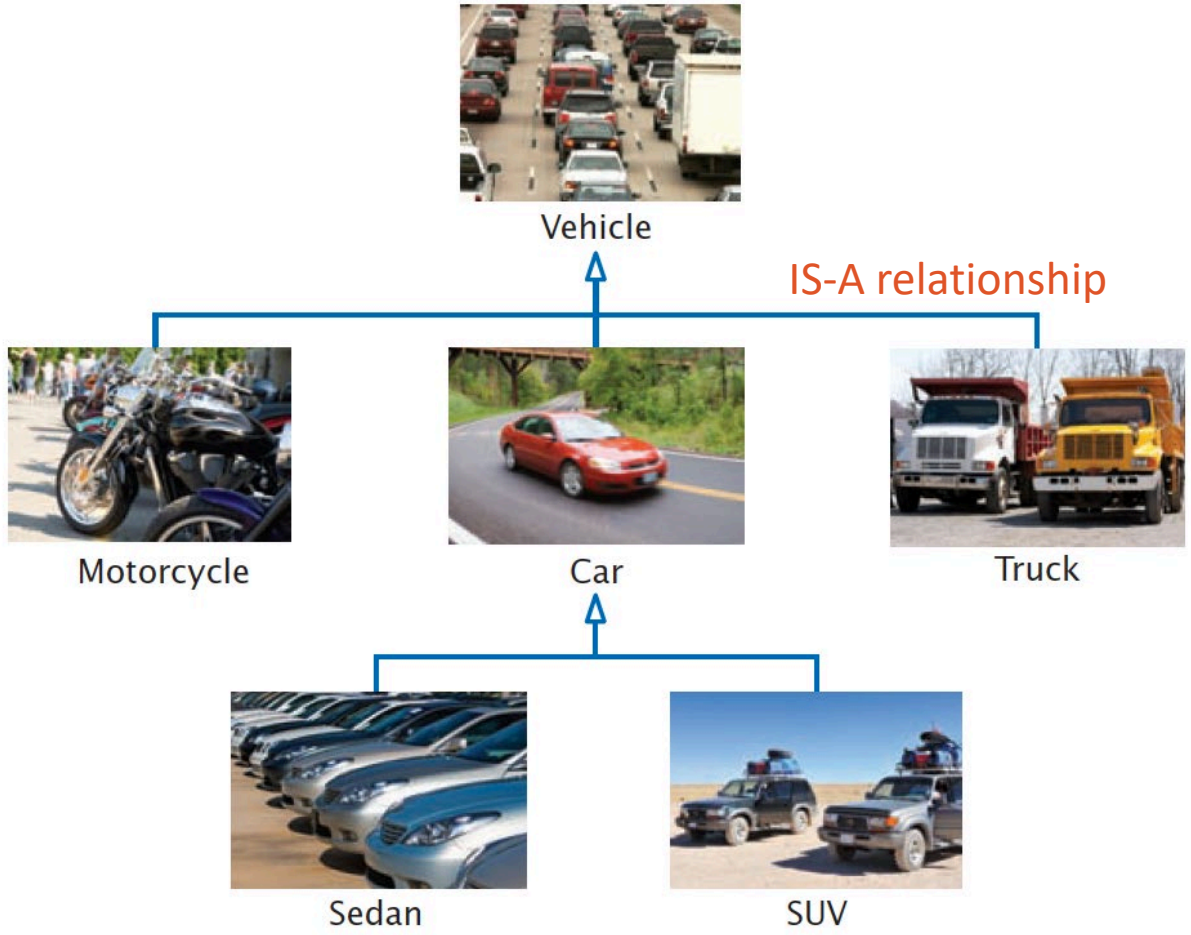
- **Polymorphism**
  - We will explain how to work in the same manner with different objects, which define a specific implementation of some abstract behavior.

1. Inheritance Hierachies

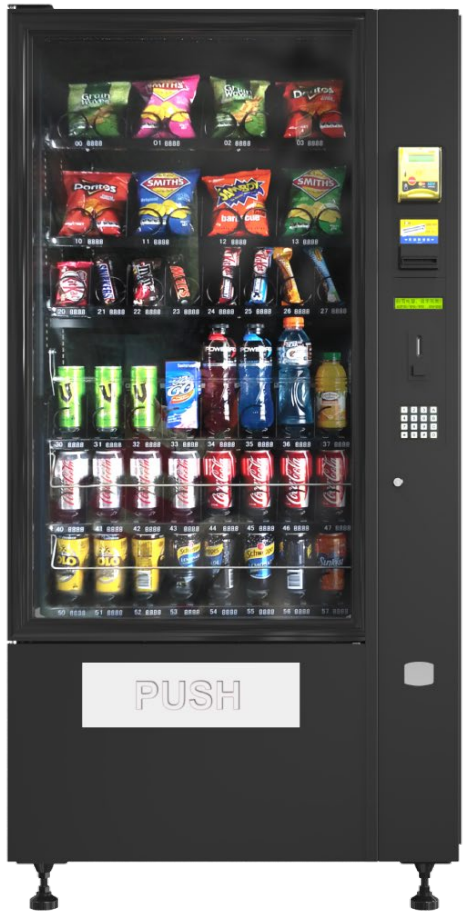
Biology Sense
Transmission of genetically controlled characteristics: the transmission of genetically controlled characteristics or qualities from parent to offspring.



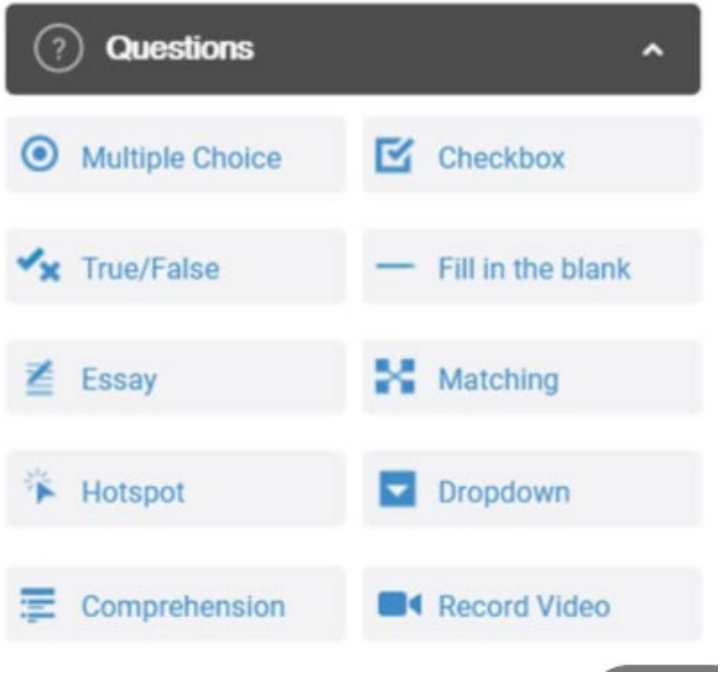
Inheritance is a relationship between a more general class (superclass) and a more specialized class (subclass).



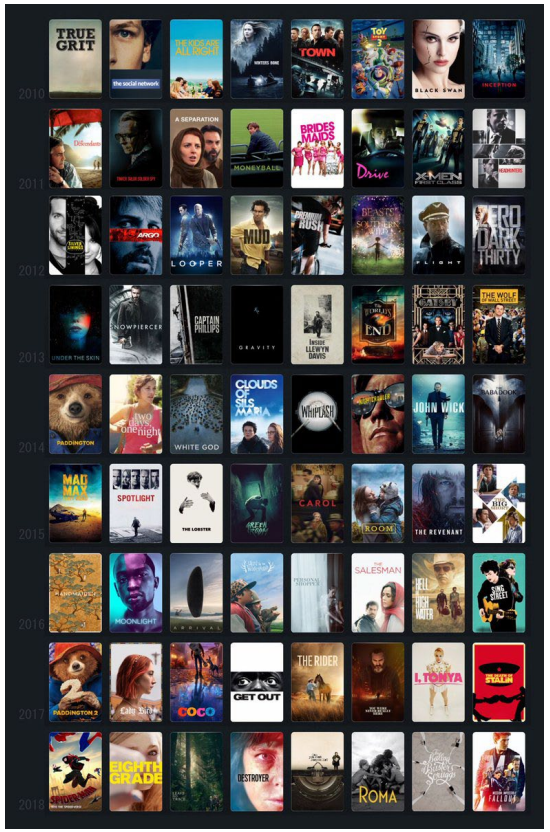
Can you comeup with superclass/subclasses



Creating a quiz for students



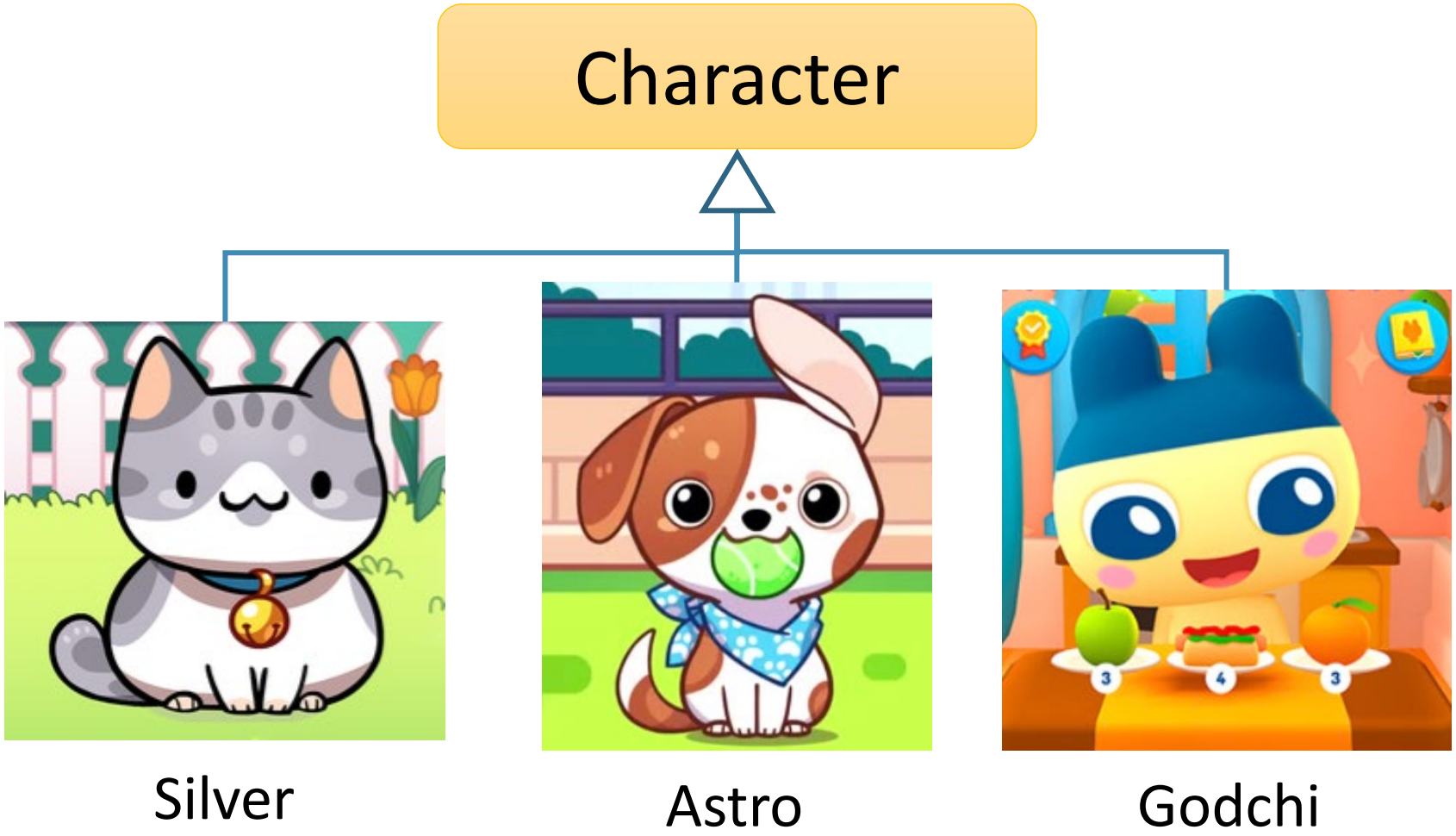
Streaming Contents



Why we need inheritance?

- To reuse code instead of duplicating it
- Usually in two forms
  1. A **subclass** inherits the **methods** of the **superclass**.
     For example, if the `Vehicle` class has a `drive` method, then a subclass `Car` automatically inherits that method. No need to rewrite the same code.
  2. Reuse algorithms that manipulate `Vehicle` objects. Because a car is a special kind of vehicle, we can use `Car` object in such an algorithm.

Let’s look at this scenairo – Reuse #1



We make a game to play with a variety of characters. Suppose all of them can

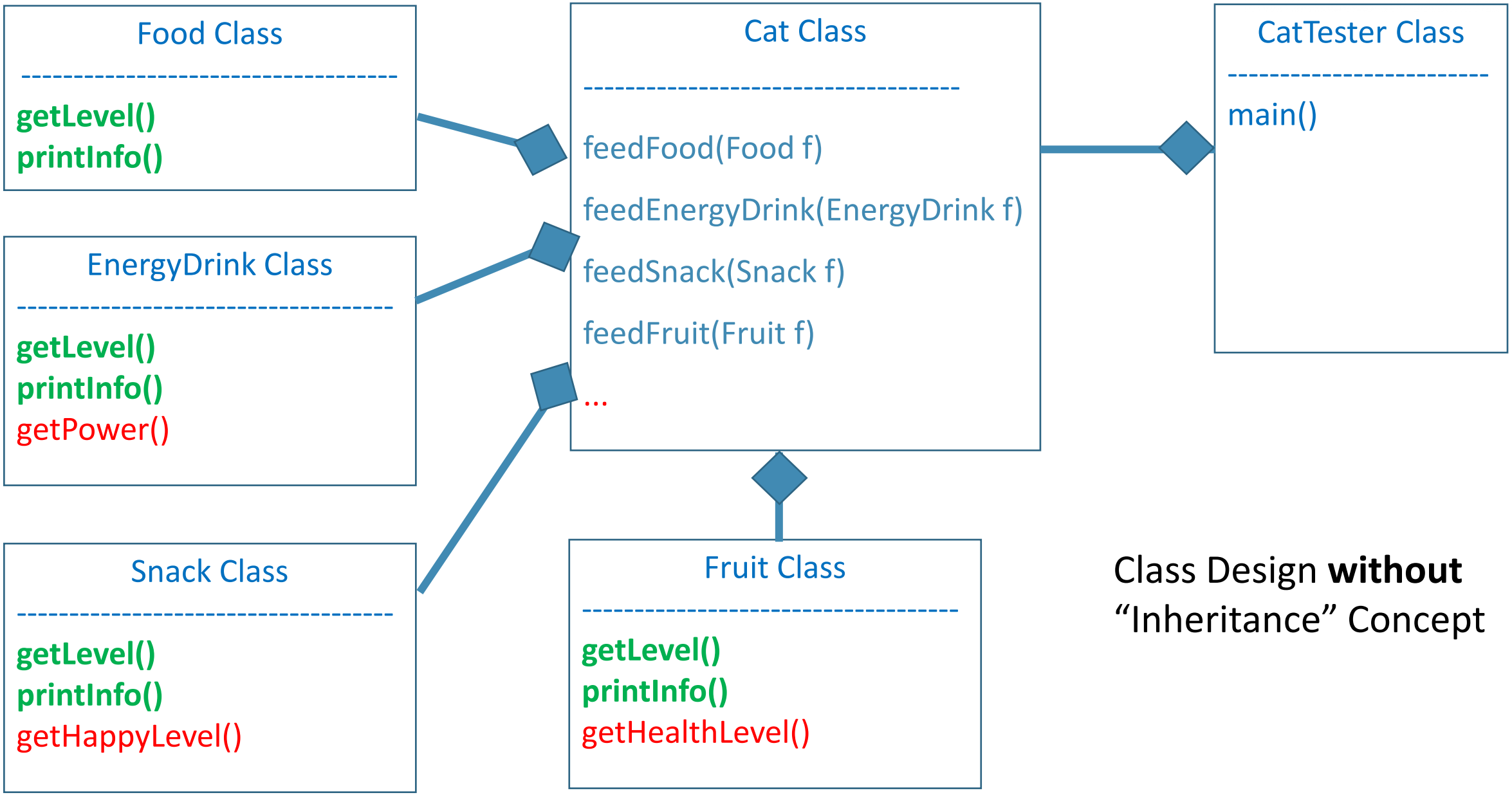
```
talk()
move()
eat()
```

Let’s look at this scenairo – Reuse #2

Name: Silver
Color: gray
Hungry Level: 8
Energy Level: 10
Happy Level: 3

How to feed Sliver with variety of food that he loves; e.g., fish, cookie, energy drink, grape

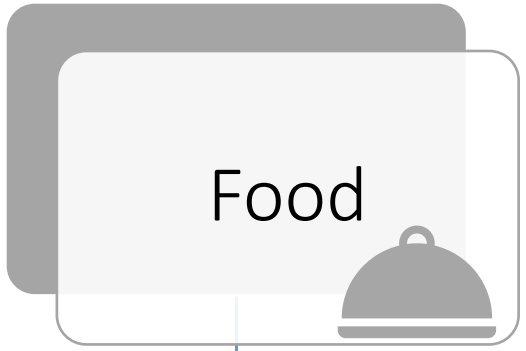
Do we need to create different methods for different kinds of food?



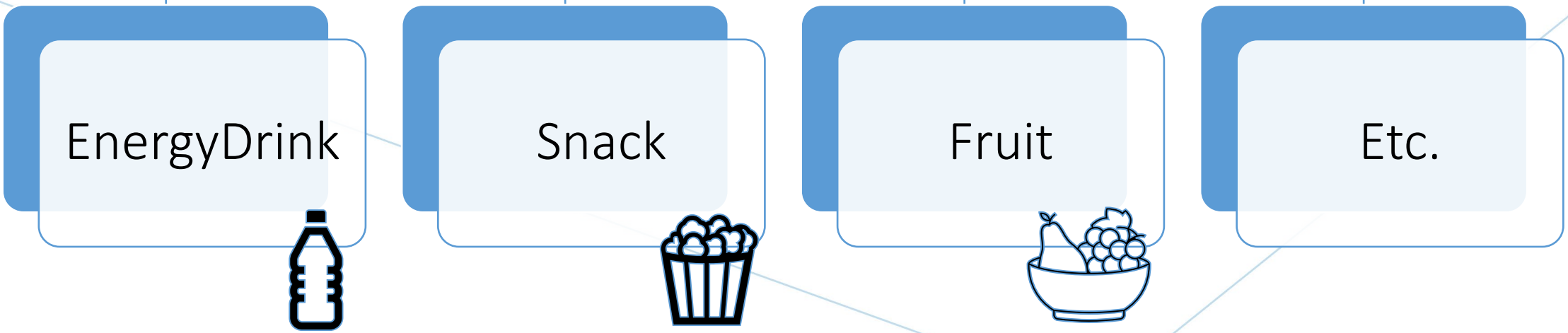
Class Design **without** "Inheritance" Concept

Inheritance Hierachy for Food Class

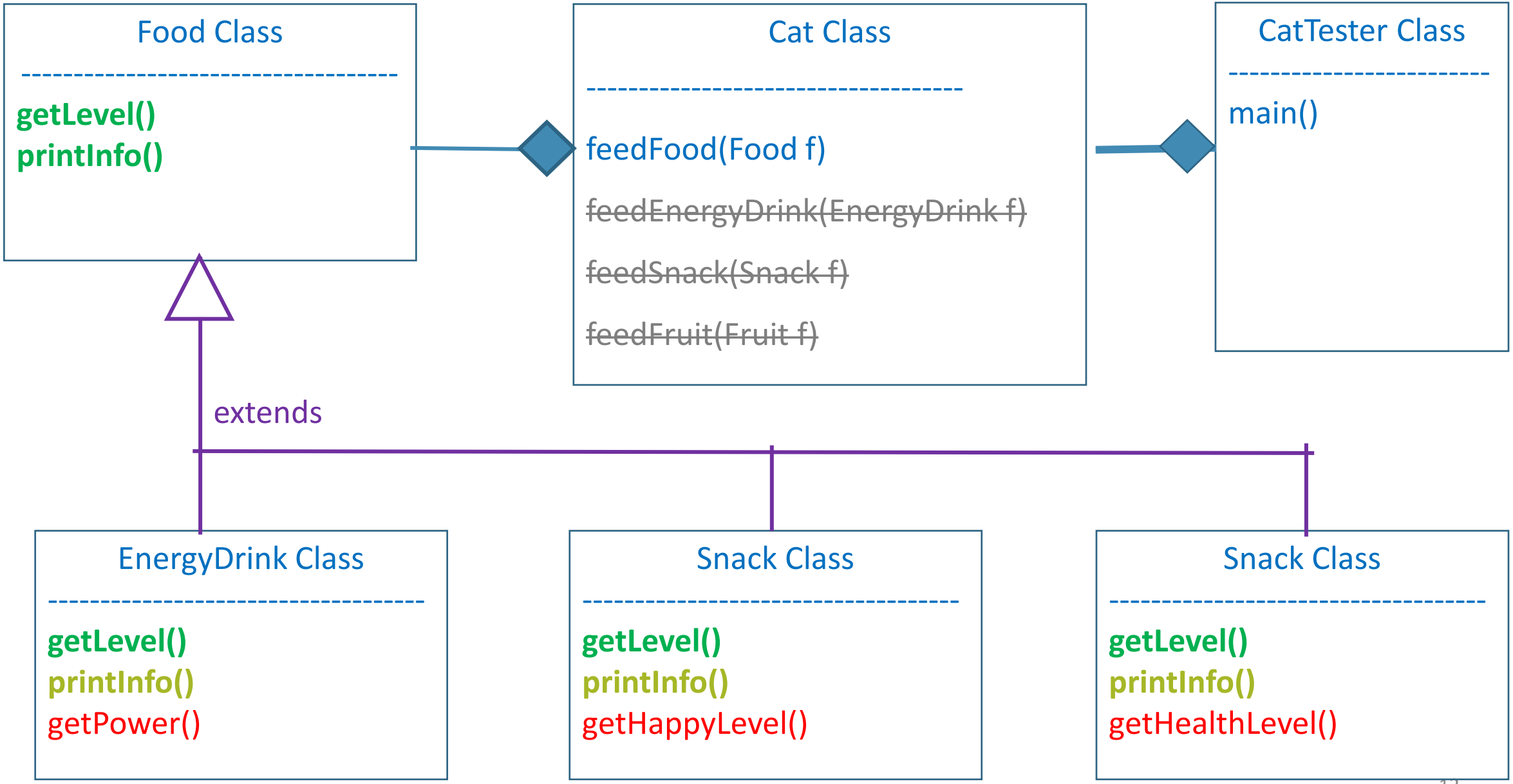
Superclass



Subclasses



New Class Design **with** “Inheritance” Concept



Food Class

```
name:
level:
---------------
Food(name, level)
printInfo()
getLevel()
```

Which instance fields of these two classes are the same?

Which methods have the same name and behavior?

Which methods have the same name but different behavior?

EnergyDrink Class

```
name:
level:
power:
---------------
EnergyDrink(name, level, power)
printInfo()
getLevel()
getPower()
```

Which methods are not the same?

How about the constructor? Same or different way to create?

2. Implementing Subclass

- Subclass inherit superclass by adding "**`extends`**" keyword.
- Subclass only include what makes the subclass **different form** its superclass
- Subclass objects automatically have the **instance variables** that are declared in the superclass. So you only declare instance variables that are not part of the superclass. (e.g., `power` in EnergyDrink)
- Subclass objects can call all inherited method from the superclass. You only implement any specialized method for the subclass. For example, EnergyDrink has `getPower()` method

Syntax

```
public class SubclassName extends SuperclassName
{
   instance variables
   methods
}
```

The reserved word `extends` denotes inheritance.

Instance Fields & Constructor Methods


```java
public class Food {
    public String name;
    private int level;

    Food(String name, int level){
        this.name = name;
        this.level = level;
    }

    void printInfo(){
        System.out.println(
                name + ": " + level);
    }

    int getLevel(){
        return level;
    }
}
```

```java
//EnergyDrink is a subclass, and Food is a superclass
public class EnergyDrink extends Food{
    private int power;

    EnergyDrink(String name, int level, int power){
        // call constructor method of the superclass
        super(name, level);

        // assign a input value to a new variable (power)
        this.power = power;
    }
}
```

How to create subclass's **constructor** method.

A subclass constructor can only initialize the subclass instance variables.

But the superclass instance variables also need to be initialized.

-> via **superclass constructor**, use the **super** reserved word in the ***first statement*** of the subclass constructor.

Syntax

```
public ClassName(parameterType parameterName, . . .)
{
   super(arguments);
   . . .
}
```

Accessing Inherited Instance Fields


```java
public class Food {
    public String name;
    private int level;

    Food(String name, int level){
        this.name = name;
        this.level = level;
    }

    void printInfo(){
        System.out.println(
                name + ": " + level);
    }

    int getLevel(){
        return level;
    }
}
```

```java
//EnergyDrink is a subclass, and Food is a superclass
public class EnergyDrink extends Food{
    private int power;

    EnergyDrink(String name, int level, int power){
        // call constructor method of the superclass
        super(name, level);

        // assign a input value to a new variable (power)
        this.power = power;
    }

    // override method
    void printInfo(){
        System.out.println(
                super.name + ": " + getLevel() + ", power: " + power);
    }
```

The instance variable "name" inherited from superclass is **public**, so the subclass can access it directly using `super.name` or just `name`.

But, "`level`" inherited from superclass is **private**, this subclass cannot access it directly. You have to get its value by calling `getLevel()` method.

Shadowing Instance Field (DON'T do this)

```java
public class Food {
    public String name;
    private int level;

    Food(String name, int level){
        this.name = name;
        this.level = level;
    }

    void printInfo(){
        System.out.println(
                name + ": " + level);
    }

    int getLevel(){
        return level;
    }
}
```

```java
//EnergyDrink is a subclass, and Food is a superclass
public class EnergyDrink extends Food{
    private int power;
    private int level;

    EnergyDrink(String name, int level, int power){
        // call constructor method of superclass
        super(name, level);

        // assign a input value to a new variable (power)
        this.power = power;
    }
    // override method
    void printInfo(){
        System.out.println(
                super.name + ": " + level + ", power: " + power);
    }
```

```
name: M-100
level: 1 (inherit from Food)
power: 10
level: 1 (new variable in EnergyDrink)
```

Inherit, Override, New Methods

```java
public class Food {
    public String name;
    private int level;

    Food(String name, int level){
        this.name = name;
        this.level = level;
    }

    void printInfo(){
        System.out.println(
                name + ": " + level);
    }

    int getLevel(){
        return level;
    }
}
```

```java
//EnergyDrink is a subclass, and Food is a superclass
public class EnergyDrink extends Food{
    private int power;

    EnergyDrink(String name, int level, int power){
        // call constructor method of the superclass
        super(name, level);

        // assign a input value to a new variable (power)
        this.power = power;
    }

    // override method
    void printInfo(){
        System.out.println(
                super.name + ": " + getLevel() + ", power: " + power);
    }

    // new method (doesn't have this in Food)
    int getPower(){
        return power;
    }
}
```

3. Overriding Methods

- The subclass inherits all the mehods from the superclass.
- However, if you are not satisfied with the behavior of an inherited method, you can **override** it.
- As shown in previouse slide, we would like to display power level of the energy drink when we print it out. So we need to override the `printInfo` method to include power information.

Now, both Food and EnergyDrink classes have `printInfo()` method (overridden method)

How to call `printInfo()` method of the superclass inside the subclass? --> `super.printInfo()`

```
public class EnergyDrink extends Food{
    private int power;

    EnergyDrink(String name, int level, int power){
        // call constructor method of the superclass
        super(name, level);
        // assign a input value to a new variable (power)
        this.power = power;
    }

    // override method
    void printInfo(){
        printInfo();
        System.out.println("power: " + power);
    }

    // new method (doesn't have this in Food)
    int getPower(){
        return power;
    }
}
```

Food Class

```
void printInfo(){
    System.out.println(
            name + ": " + level);
}
```

EnergyDrink Class

```
// override method
void printInfo(){
    super.printInfo();
    System.out.println("power: " + power);
}
```

super is simply a reserved word that forces execution of the superclass method.

OUTPUT

```
M-100: 1
power: 10
```

4. More Example

Java Swing Hierarchy

- Superclass `JComponent` has methods `getWidth, getHeight`
- `AbstractButton` class has methods to set/get button text and icon

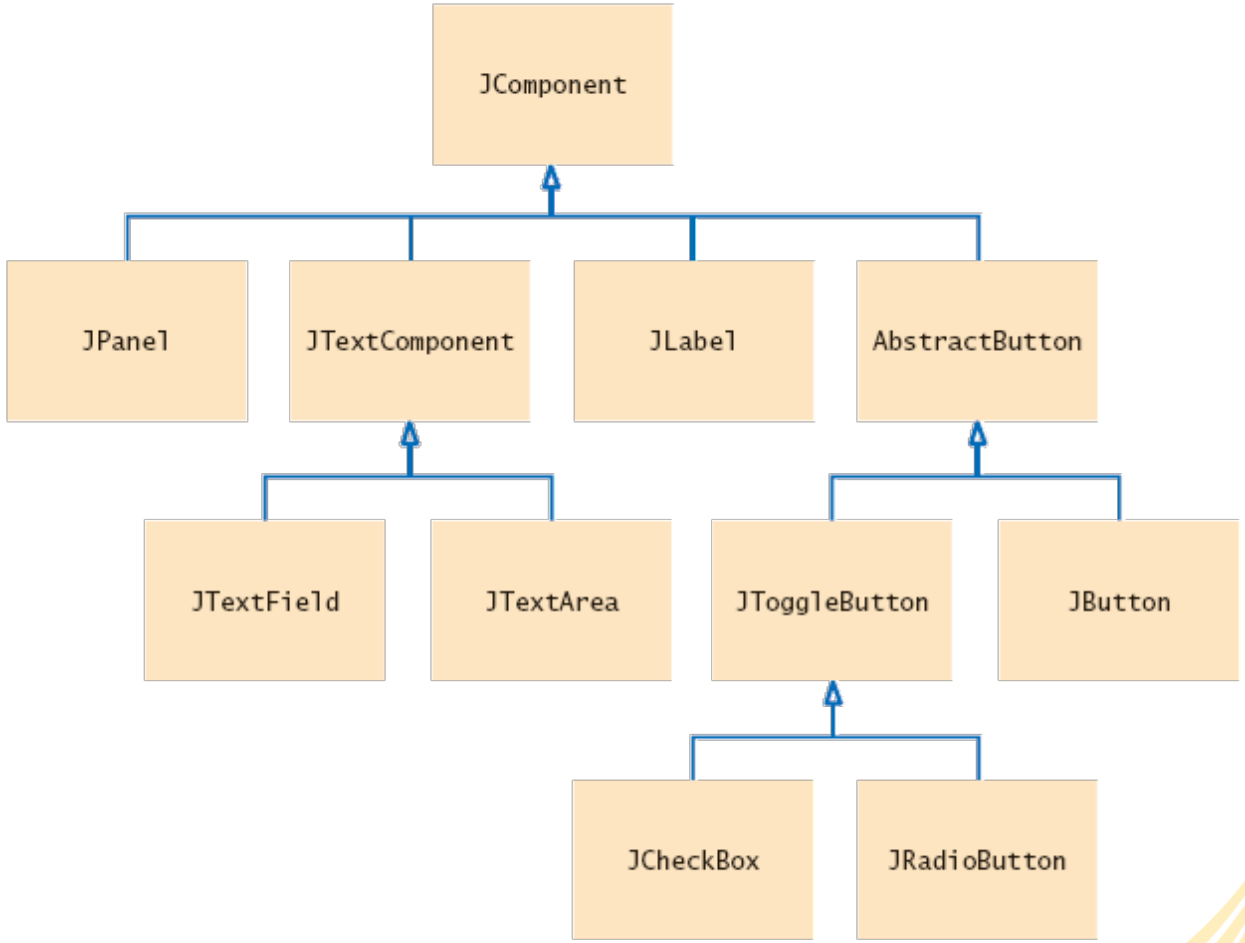
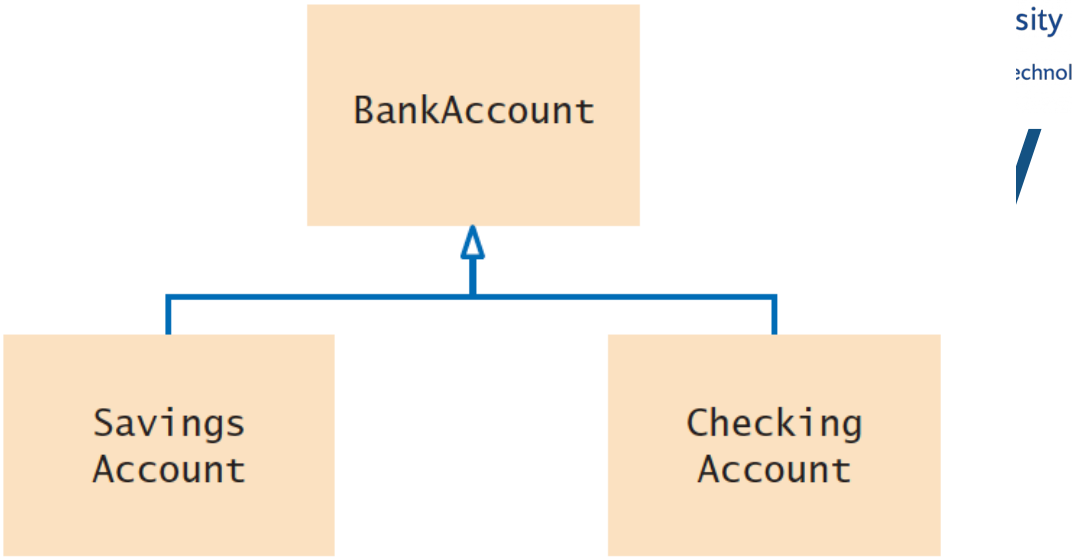


Figure 4 A Part of the Hierarchy of Swing User Interface Components

BankAccount



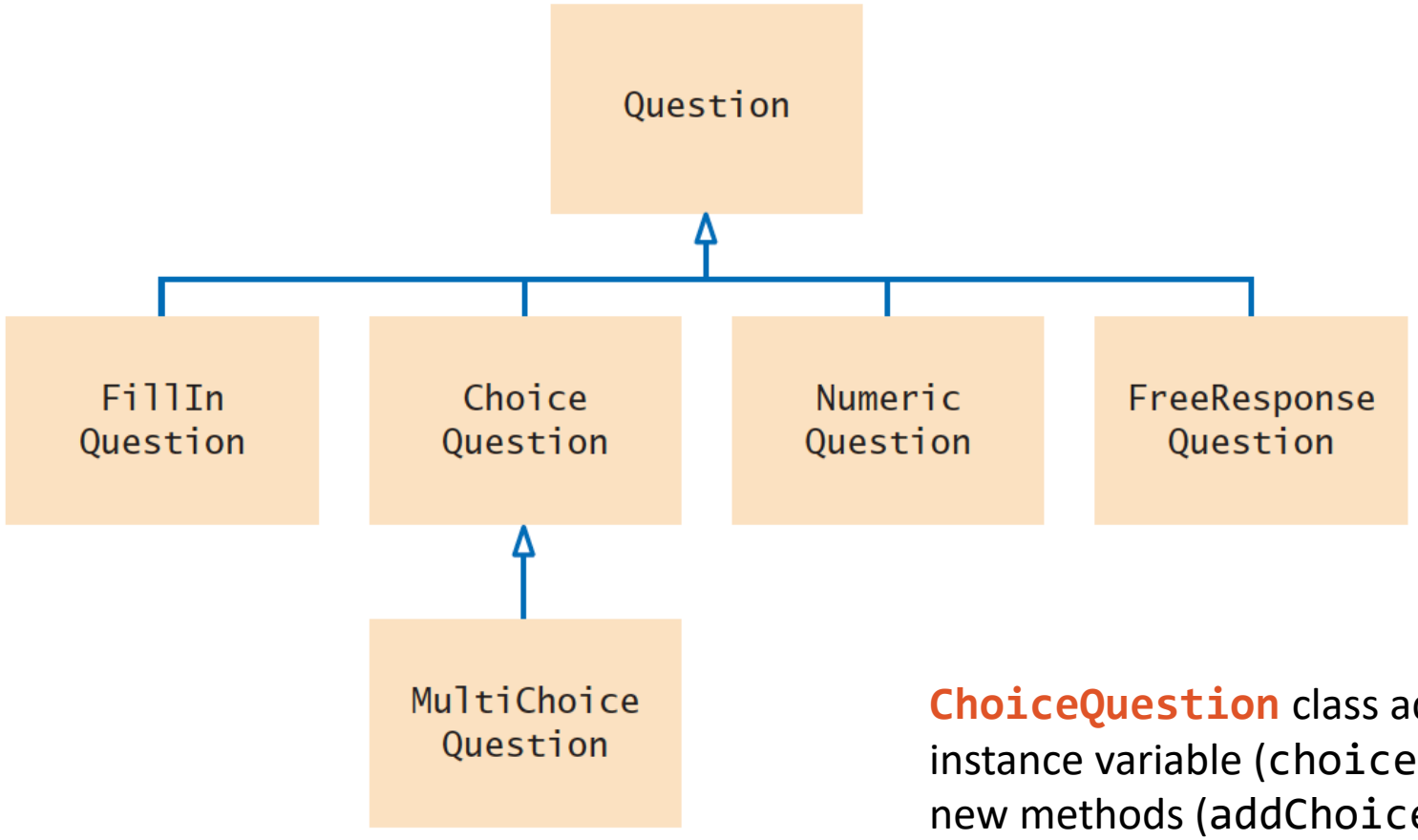
General Characteristics

current balance
owner
opened date

deposit (…)
withdraw (…)
transfer (…)

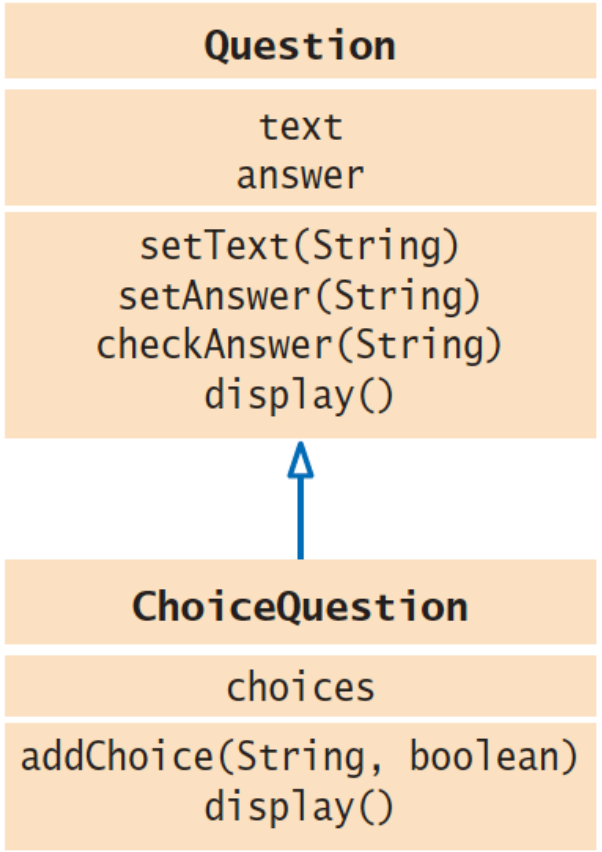
	Saving Account	**Checking Account**
Withdrawal Restrictions	Typically 3-6 withdrawals a month	None
Designed for	Saving money risk-free	Regular use
Interest Earned	Yes	Nominal/None
Debit Card	No	Yes
Bill pay	No	Yes
Overdraft	No	Yes

Questions



ChoiceQuestion class adds an instance variable (choices) and new methods (addChoice), and overrides a method (display)

ChoiceQuestion class extends Question lass



Syntax

```
public class SubclassName extends SuperclassName
{
   instance variables
   methods
}
```

The reserved word extends denotes inheritance.

Declare instance variables that are added to the subclass.

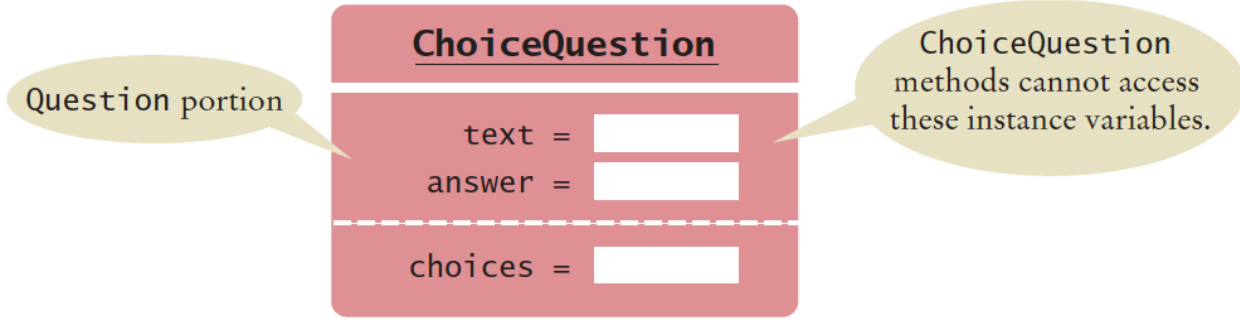
Declare methods that are added to the subclass.

Declare methods that the subclass overrides.

Subclass

Superclass

```
public class ChoiceQuestion extends Question
{
   private ArrayList<String> choices;

   public void addChoice(String choice, boolean correct) { . . . }

   public void display() { . . . }
}
```



5. Special Topics

- Final Methods and Classes
- Protected Access
- Converting between subclass and superclass
  - Using object casting `(class_name)` and `instanceof` keyword
- Type of Inhertiance

5.1 Final Methods and Classes

- You may want to do *opposite* from extends and *prevent* other programmers from creating subclasses of your class or from overriding certain methods.
- You can use the **`final`** reserved word.
- For example, the **`String`** class in the Java library has bee declared as

```
public final class String { . . . } // no class can extend String class
```

- Another example, you can declare individual method as final:

```
public final int getLevel() { . . . } // subclass cannot override this method
```

You cannot cheat the program by override this mehtod and return infinite food level!!!

5.2 Protected Access

- With **`private`** access, the subclass *cannot* directly access superclass instance variables.
- To allow **only subclasses** but not other classes to access, you can use **`protected`**.

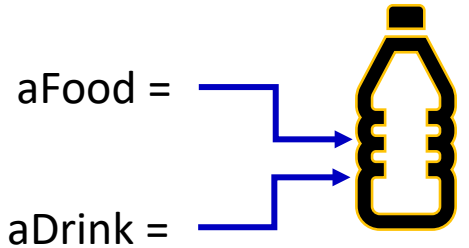
```
public class Food {
  public String name;
  protected int level;

  Food(String name, int level){
    this.name = name;
    this.level = level;
  }

  void printInfo(){
    System.out.println(
        name + ": " + level);
  }
}
```

```
public class EnergyDrink extends Food{
  private int power;

  // override method
  void printInfo(){
    System.out.println(
        name + ": " + level + ", power: " + power);
  }

}
```

5.3 Converting betwee subclass/superclass

- It is legal to store a subclass reference in a superclass variable

```
Food aFood = new EnergyDrink("M-150", 3, 9);
```



- Occationally, we might need to convert from superclass to subclass

```
EnergyDrink aDrink = (EnergyDrink) food;
```

- This cast is somewhat dangerous.
  If the object "f" is not actually refers to the subclass "EnergyDrink", your program will fail.

  So we should check before casting!!!

Syntax *object* `instanceof` *TypeName*

If anObject is null, instanceof returns false.

Returns true if anObject can be cast to a Question.

The object may belong to a subclass of Question.

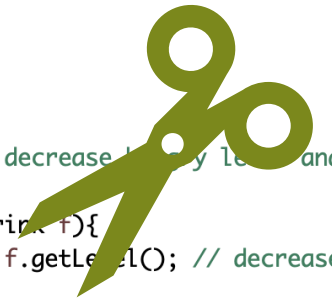
```
if (anObject instanceof Question)
{
   Question q = (Question) anObject;
   . . .
}
```

You can invoke Question methods on this variable.

Two references to the same object.

Now, we can remove feedFood(EnergyDrink f) method in Cat with no syntax error

```
/*
 * If the hungry level is less than 0, set its value to 0
 */
void feedFood(Food f){
    hungry = hungry - f.getLevel(); // decrease hungry
    if(hungry < 0){
        hungry = 0;
    }
}

/*
 * Feed EnergyDrink - decrease hungry level and increase energy level too
 */
void feedFood(EnergyDrink f){
    hungry = hungry - f.getLevel(); // decrease hungry
    if(hungry < 0){
        hungry = 0;
    }
    // increase energy level
    energy = energy + f.getPower();
}
```



```
System.out.println("------------------------");
System.out.println("After playing");
System.out.println("Energy Level: " + c1.getEnergy());  // 8
System.out.println("Hungry Level: " + c1.getHungry());  // 17

// create a Food object named f1
Food f1 = new Food("Fish", 5);
f1.printInfo();

// create an EnergyDrink object named d1
EnergyDrink d1 = new EnergyDrink("M-100", 1, 10);
d1.printInfo();

c1.feedFood(f1);    // Fish
c1.feedFood(d1);    // M-100
System.out.println("------------------------");
System.out.println("After feeding");
System.out.println("Energy Level: " + c1.getEnergy());  // expected: 18
System.out.println("Hungry Level: " + c1.getHungry());  // expected: 11
```

BUT the energy level after feeding is WRONG!!!
We expected the energy level to be 18, but we got only 8. Why?

To fix the problem and avoid creating a new method in Cat class, we can do this

instanceof and (object) casting

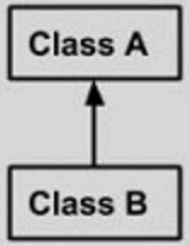
```
/*
 * If the hungry level is less than 0, set its value to 0
 */
void feedFood(Food f){
    hungry = hungry - f.getLevel(); // decrease hungry
    if(hungry < 0){
        hungry = 0;
    }
}

/*
 * Feed EnergyDrink - decrease hungry level and increase energy level too
 */
void feedFood(EnergyDrink f){
    hungry = hungry - f.getLevel(); // decrease hungry
    if(hungry < 0){
        hungry = 0;
    }
    // increase energy level
    energy = energy + f.getPower();
}
```

```
/*
 * If the hungry level is less than 0, set its value to 0
 */
void feedFood(Food f){
    hungry = hungry - f.getLevel(); // decrease hungry
    if(hungry < 0){
        hungry = 0;
    }
    if(f instanceof EnergyDrink){
        energy = energy + ((EnergyDrink) f).getPower();
    }
}
```

If the input parameter is an EnergyDrink, we will increase the energy level by the power level of that food. Now the output is correct!

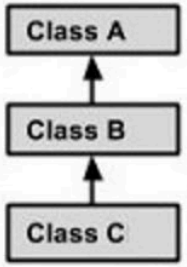
5.4 Type of Inhertiance

Single Inheritance



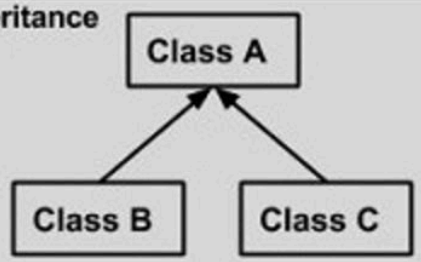

```
public class A {
   .....
}
public class B extends A {
   .....
}
```

Multi Level Inheritance



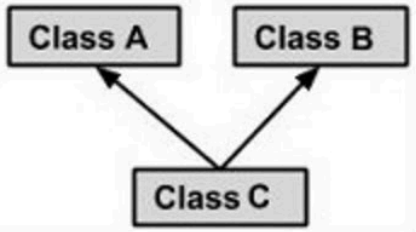

```
public class A { ..................}
public class B extends A {.....................}
public class C extends  B {..................... }
```

Hierarchical Inheritance




```
public class A { ..................}
public class B extends A {.....................}
public class C extends A {..................... }
```

Multiple Inheritance




```
public class A { ..................}
public class B {.....................}
public class C extends  A,B {
   .....................
} // Java does not support mutiple Inheritance
```

Appendix – Cat & Food

Mahidol University
Faculty of Information and Communication Technology

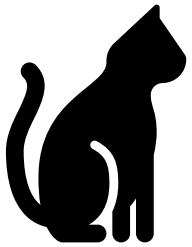
object

c1 =

```
name: Silver
color: gray
hungry: 10
---------------
greeting()
```

c2 =

```
name: Elsa
color: white
hungry: 5
---------------
greeting()
```



```
name:
color:
hungry:
----------------
greeting()
```

```
System.out.println("Meow! " + this.name);
```

1) Simplest Cat Class

```java
public class Cat {
    /*
     * Instance fields or Instance variables or Attributes
     */
    String name;
    String color;
    int hungry;

    /*
     * When there is no constructor method provided,
     * the class has a default constructor
     * with empty parameter and empty body.
     * For example, this Cat class has this constructor
     *
     * public Cat(){
     * }
     */

    /*
     * Instance method with no return (void)
     * To display the greeting message
     */
    void greeting(){
        System.out.println(
         "Meow! " + this.name);
    }
}
```

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

        Cat c1 = new Cat(); // Create Cat object via Cat()
                            // default constructor method

        c1.name = "Silver"; // Assign value to its attribute
        c1.color = "gray";  // Assign value to its attribute
        c1.hungry = 10;     // Assign value to its attribute

        Cat c2 = new Cat(); // Create another Cat object
        c2.name = "Elsa";
        c2.color = "White";
        c2.hungry = 5;

        c1.greeting();      // call greeting() method by c1
        c2.greeting();      // call greeting() method by c2
    }
}
```

OUTPUT:

Meow! Silver
Meow! Elsa

2) Cat Class with Constructor Method

```java
public class Cat {
    String name;
    String color;
    int hungry;

    /*
     * Define a constructor method with 3 parameters
     */
    Cat(String name, String color, int hungry){
        this.name = name;       // "this." is required because
                                // the names of the parameter and
                                // the attribute are the same.

        this.color = color;     // "this." is required
        this.hungry = hungry;   // "this." is required
    }

    void greeting(){
        System.out.println(
         "Meow! " + this.name); // "this." is no required
                                // since no duplicate variable name
    }

}
```

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

        // Create Cat object via defined constructor method
        Cat c1 = new Cat("Silver", "gray", 10);
        Cat c2 = new Cat("Elsa", "white", 5);

        /*
         * NOTE: If you already define your own constructor, then
         * the default constructor method is no longer valid.
         * Cat c = new Cat(); // error!!!
         */

        c1.greeting();
        c2.greeting();
    }
}
```

OUTPUT:

Meow! Silver
Meow! Elsa

3) Public vs Private

```java
public class Cat {
    String name;            // public by default
    public String color;    // public access modifier
    private int hungry;     // private access modifier

    /*
     * Constructor method
     */
    Cat(String name, String color, int hungry){
        this.name = name;
        this.color = color;
        this.hungry = hungry;
    }

    /*
     * Instance method
     */
    void greeting(){
        System.out.println(
         "Meow! " + this.name);
    }

    /*
     * Getter method with return type (int)
     */
    int getHungry(){
        return hungry;
    }
}
```

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

        Cat c1 = new Cat("Silver", "gray", 10);

        // access "public" instance field
        // using objectname.instancefiledname
        // correct... yea!
        System.out.println(c1.name);
        System.out.println(c1.color);

        // access "private" instance field
        // error... boo!
        //System.out.println(c1.hungry);

        // have to access through the getter method
        // correct... yea!
        System.out.println(c1.getHungry());
    }
}
```

4) static variable/method & final

```java
public class Cat {
    public String name, color;
    private int hungry;

    // "static" keyword to indicate a variable of the class
    static int objCounter = 0;

    // "final" keyword to indicate an unchangeable variable
    final String TYPE = "cat";

    // static and final are commonly used together
    static final int LIVE = 9;

    Cat(String name, String color, int hungry){
        this.name = name;
        this.color = color;
        this.hungry = hungry;
        this.objCounter++;      // increase the counter when
                                // an object is created
    }

    // "static" keyword to indicate a method of class
    public static void begin(){
        System.out.println("Let's start");
    }

    void greeting(){
        System.out.println("Meow! " + this.name);
    }
}
```

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

        // Access static variable using Cat class
        System.out.println(Cat.objCounter);     // OUTPUT: 0

        Cat c1 = new Cat("Silver", "gray", 10);
                // this constructor will increase objCounter
        System.out.println(Cat.objCounter);     // OUTPUT: 1

        Cat c2 = new Cat("Elsa", "white", 5);
        System.out.println(Cat.objCounter);     // OUTPUT: 2

        // Call static method using Cat class
        Cat.begin();        // OUTPUT: Let's start

        // Call instance method using specific object
        c1.greeting();      // OUTPUT: Meow! Silver
        c2.greeting();      // OUTPUT: Meow! Elsa

        // Access final variable
        System.out.println(c1.TYPE);    // OUTPUT: cat

        // Access static final variable
        System.out.println(Cat.LIVE);   // OUTPUT: 9

    }
}
```

5) Method with Parameters

```java
public class Cat {
    public String name;
    public String color;
    private int hungry;

    Cat(String name, String color, int hungry){
        this.name = name;
        this.color = color;
        this.hungry = hungry;
    }

    /*
     * instance method with one parameter (primitive type)
     * This method changes the hungry level (instance field)
     */
    void feed(int level){
        hungry = hungry - level;
    }

    void greeting(){
        System.out.println("Meow! " + this.name);
    }

    int getHungry(){
        return this.hungry;
    }
}
```

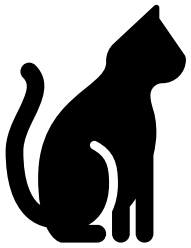
```java
public class CatTester {
    public static void main(String[] args){

        Cat c1 = new Cat("Silver", "gray", 10);
        Cat c2 = new Cat("Elsa", "white", 5);

        // to display hungry level
        System.out.println(c1.getHungry());     // OUTPUT: 10

        // feed c1 object which only effects its hungry level
        c1.feed(2);

        // to display hungry level after feed(x)
        System.out.println(c1.getHungry());     // OUTPUT: 8

        // hungry level of c2 is not effected
        System.out.println(c2.getHungry());     // OUTPUT: 5
    }
}
```

c1 =

```
name: Elsa
color: white
hungry: 15
---------------
Cat("Elsa", "white", 15)
display()
…
FeedFood(food)
```

f1 =

```
name: fish
level: 5
---------------
Food("fish", 5)
printInfo()
getLevel()
```



Cat Class

```
name:
color:
hungry:
---------------
Cat(name, color, hungry)
display()
greeting()
getHungry()
feedFood(food)
```

Food Class

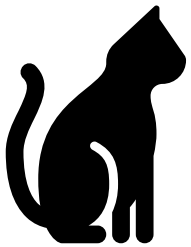
```
name:
level:
---------------
Food(name, level)
printInfo()
getLevel()
```

6) Working with Multiple Classes – Food!

```java
public class Food {
    private String name; // instance field
    private int level;   // instance field

    // Constructor method
    Food(String n, int lev){
        this.name = n;
        this.level = lev;
    }

    // Display food information
    void printInfo(){
        System.out.println(
                name + ":" + this.level);
    }

    // Get hungry's level this food
    // can decrease
    int getLevel(){
        return this.level;
    }
}
```

```java
public class Cat {
    public String name, color;
    private int hungry;

    Cat(String name, String color, int hungry){
        this.name = name;
        this.color = color;
        this.hungry = hungry;
    }

    void feed(int level){
        hungry = hungry - level;
    }

    /*
     * Instance method with one parameter (object type)
     * It changes the hungry level based on
     * the level score of the food object "f"
     */
    void feedFood(Food f){
        hungry = hungry - f.getLevel();
    }

    int getHungry(){
        return this.hungry;
    }
}
```

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

        Cat c1 = new Cat("Silver", "gray", 10);

        System.out.println(c1.getHungry()); // OUTPUT: 10
        c1.feed(2);
        System.out.println(c1.getHungry()); // OUTPUT: 8

        // Create Food object whose name is fish
        // and level is 3
        Food f1 = new Food("Fish", 5);
        System.out.println(f1.getLevel());  // OUTPUT: 5

        c1.feed(f1.getLevel());
            // since f1.getLevel() is equal to 5
            // so this code is similar to c1.feed(5);
        System.out.println(c1.getHungry()); // OUTPUT: 3

        // Let's work with another object c2
        Cat c2 = new Cat("Elsa", "white", 5);

        System.out.println(c2.getHungry()); // OUTPUT: 5
        c2.feedFood(f1);
            // Food object f1 is passed as parameter
        System.out.println(c2.getHungry()); // OUTPUT: 0
    }
}
```



Cat Class

```
name:
color:
hungry:
energy:
---------------
Cat(name, color, hungry, energy)
display()
greeting()
getHungry()
getEnergy()
play()
sleep()
feedFood(food)
```

Besides hungry level, let's say the Cat also has **energy level**.

This energy level will be reduced when the Cat plays, but it will be increased when the Cat sleeps or eats supplementary food that boost the energy such as energy drink

Updated Cat Class

```java
public class Cat {
    public String name, color;
    private int hungry, energy;     // Add another attribute (energy)

    Cat(String name, String color, int hungry, int energy){
        this.name = name;
        this.color = color;
        this.hungry = hungry;
        this.energy = energy;
    }

    /*
     * show Cat's info
     */
    void display(){
        System.out.println("------------------------");
        System.out.println("Name: " + this.name);
        System.out.println("Color: " + this.color);
        System.out.printf("Hungry (%d), Energy (%d)\n", hungry, energy);
    }

    /*
     * greeting method()
     */
    void greeting(){
        System.out.println("Meow! I'm " + this.name);
    }

    int getHungry(){
        return this.hungry;
    }

    int getEnergy(){
        return this.energy;
    }

    /*
     * play() method
     * Increase hungry level, and decrease energy level
     */
    void play(){
        hungry++;
        energy--;
    }

    /*
     * sleep() method
     * Increase both energy and hungry level
     */
    void sleep(){
        hungry++;
        energy++;
    }

    /*
     * If the hungry level is less than 0, set its value to 0
     */
    void feedFood(Food f){
        hungry = hungry - f.getLevel(); // decrease hungry
        if(hungry < 0){
            hungry = 0;
        }
    }
```

EnergyDrink Class

```
name:
level:
power:
---------------
Food(name, level, power)
printInfo()
getLevel()
getPower()
```

d1 =

```
name: M-100
level: 1
power: 10
---------------
Food("M-100", 1, 10)
printInfo()
getLevel()
getPower()
```

```java
public class EnergyDrink {
    private String name;    // same attribute as Food
    private int level;      // same attribute as Food
    private int power;

    EnergyDrink(String name, int level, int power){
        this.name = name;
        this.level = level;
        this.power = power;
    }

    // same method as Food
    void printInfo(){
        System.out.println(
                name + ": " + level + ", power: " + power);
    }

    // same method as Food
    int getLevel(){
        return level;
    }

    // new method (doesn't have this in Food)
    int getPower(){
        return power;
    }
}
```

EnergyDrink Class
WITHOUT INHERITANCE

Overall Source Code of Food and EnergyDrink

WITH INHERITANCE


```java
public class Food {
    public String name;
    private int level;

    Food(String name, int level){
        this.name = name;
        this.level = level;
    }

    void printInfo(){
        System.out.println(
                name + ": " + level);
    }

    int getLevel(){
        return level;
    }
}
```

```java
//EnergyDrink is a subclass, and Food is a superclass
public class EnergyDrink extends Food{
    private int power;

    EnergyDrink(String name, int level, int power){
        // call constructor method of the superclass
        super(name, level);

        // assign a input value to a new variable (power)
        this.power = power;
    }

    // override method
    void printInfo(){
        System.out.println(
                super.name + ": " + getLevel() + ", power: " + power);
    }

    // new method (doesn't have this in Food)
    int getPower(){
        return power;
    }
}
```

Instance Fields & Constructor Methods

```java
public class Food {
    public String name;
    private int level;

    Food(String name, int level){
        this.name = name;
        this.level = level;
    }

    void printInfo(){
        System.out.println(
                name + ": " + level);
    }

    int getLevel(){
        return level;
    }
}
```

```java
//EnergyDrink is a subclass, and Food is a superclass
public class EnergyDrink extends Food{
    private int power;

    EnergyDrink(String name, int level, int power){
        // call constructor method of the superclass
        super(name, level);

        // assign a input value to a new variable (power)
        this.power = power;
    }

    // override method
    void printInfo(){
        System.out.println(
                super.name + ": " + getLevel() + ", power: " + power);
    }

    // new method (doesn't have this in Food)
    int getPower(){
        return power;
    }
}
```

Inherit, Override, New Methods

```java
public class Food {
    public String name;
    private int level;

    Food(String name, int level){
        this.name = name;
        this.level = level;
    }

    void printInfo(){
        System.out.println(
                name + ": " + level);
    }

    int getLevel(){
        return level;
    }
}
```

```java
//EnergyDrink is a subclass, and Food is a superclass
public class EnergyDrink extends Food{
    private int power;

    EnergyDrink(String name, int level, int power){
        // call constructor method of the superclass
        super(name, level);

        // assign a input value to a new variable (power)
        this.power = power;
    }

    // override method
    void printInfo(){
        System.out.println(
                super.name + ": " + getLevel() + ", power: " + power);
    }

    // new method (doesn't have this in Food)
    int getPower(){
        return power;
    }
}
```