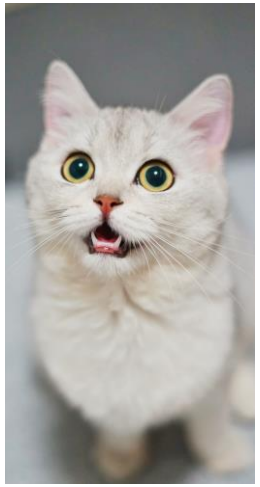
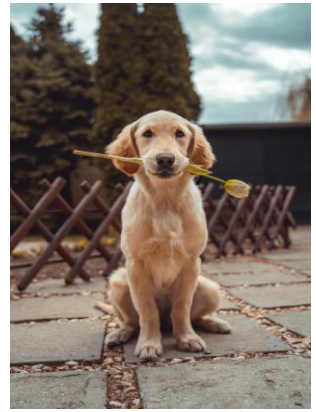
Inheritance



Class Cat
{
String color;
Int age;

Public void meow();
Public void eat();
}



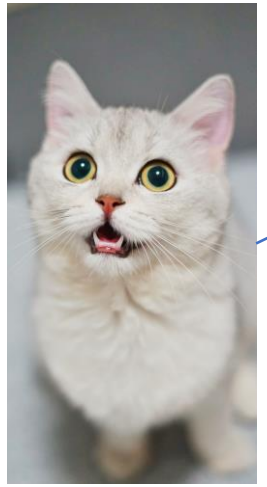
Class Dog
{
String color;
String breed;

Public void bark();
Public void eat();
}

Animal

Base Class
Parent Class
Super class
General

Is-a
instanceof

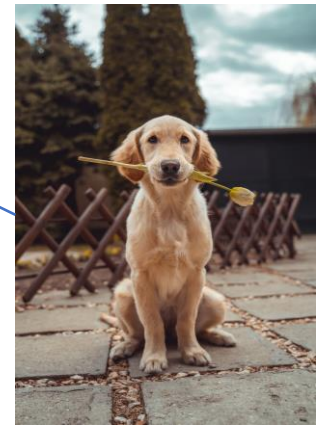


Class Animal
{
String color;

Public void eat()
}

extends

extends



Class Cat extends Animal
{
Int age;

Public void meow();
}

Code
Reusability

Derived Class
Child Class
Sub class
Specific

Class Dog extends Animal
{
String breed;

Public void bark();
}

- `System.out.println(d instanceof Animal);`
- `System.out.println(d instanceof Dog);`

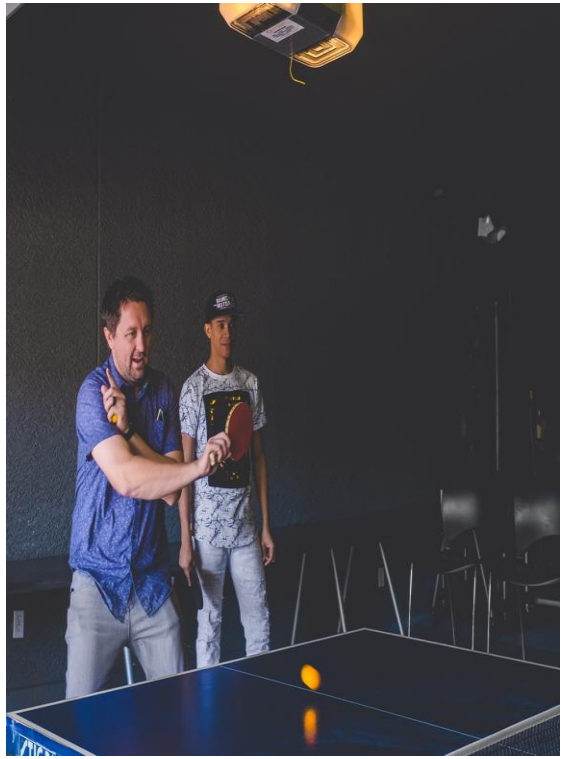
Advantages

- Saves time as methods and fields already exist
- Reduces errors as methods have been tested and used before
- Reduces the amount of new learning required to use a new class as they are already familiar with original class

Polymorphism

- Poly—Many
- Morphism-Many forms






Overloading

```
Class Dog
{
Public void bark(){System.out.println(“Woof”);}

Public void bark(int x)
{
for(int i=1;i<=x;i++)
        System.out.println(“Woof”);
}


}
```

Overriding

```
Class Dog
{
        Public void bark(){System.out.println(“Woof”);}
}
Class Hound extends Dog
{
        Public void bark(){System.out.println(“Bowl”);}
}
```

Overloading	Overrinding
Deals with multiple methods in the same class with the same name but different signatures	Deals with two methods, one in a parent class and one in a child class, that have the same signature
Lets you define a similar operation in different ways for different data	Lets you define a similar operation in different ways for different object types
compile time	runtime polymorphism

If a superclass contains:	Then its subclasses:
No constructors written by the programmer--→	Do not require constructors
A default constructor written by the programmer-→	Do not require constructors
Only non default constructors----------→	Must contain a constructor that calls the superclass constructor

ACCESS LEVELS				
Specifier	Class	Package	Subclass	Everywhere
Default	Y	Y	N	N
Private	Y	N	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Can we **override** a **static method**?

- No, we cannot **override static methods**
- **Method overriding** is based on dynamic binding at runtime and the **static methods** are bonded using **static** binding at compile time.
- Static methods are inherited but can’t be overridden

Hiding

- Parent class methods that are static are not part of a child class (although they are accessible), so there is no question of overriding it.
- When you write a new static method with the same signature, the parent static method is just hidden, not overridden.

Final Class can’t be parent

- Java’s Math class is an example of *final* class
- Can’t inherit final class
- Don’t want to corrupt methods
- String is a final class