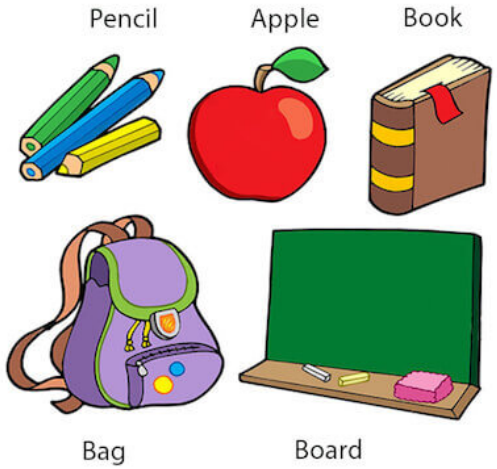
Object I

What is an OBJECT?

Object II

Objects: Real World Examples



Object in the context of Java

What Is an Object in the context of Java?

- Objects are key to understanding object-oriented technology. Look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle.
- Real-world objects share two characteristics: They all have state and behavior. Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Bicycles also have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes). Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Object in the context of Java

What Is an Object in the context of Java?

►Objects are key to understanding object-oriented technology. Look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle.

► Real-world objects share two characteristics: They all have state and behavior. Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Bicycles also have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes). Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Object in the context of Java

What Is an Object in the context of Java?

►Objects are key to understanding object-oriented technology. Look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle.

► Real-world objects share two characteristics: They all have state and behavior. Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Bicycles also have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes). Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Object Oriented Programming Concepts

► Take a minute right now to observe the real-world objects that are in your immediate area. For each object that you see, ask yourself two questions:
"What possible states can this object be in?" and "What possible behavior can this object perform?".

►For Example: your desktop lamp may have only two possible states (on and off) and two possible behaviors (turn on, turn off), but your desktop radio might have additional states (on, off, current volume, current station) and behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune). These real-world observations all translate into the world of object-oriented programming.

Object Oriented Programming Concepts

► Take a minute right now to observe the real-world objects that are in your immediate area. For each object that you see, ask yourself two questions:
"What possible states can this object be in?" and "What possible behavior can this object perform?".
►For Example: your desktop lamp may have only two possible states (on and off) and two possible behaviors (turn on, turn off), but your desktop radio might have additional states (on, off, current volume, current station) and behavior (turn on, turn off, increase volume, decrease volume, seek, scan, and tune). These real-world observations all translate into the world of object-oriented programming.

Object Oriented Programming Concepts I

Object- Unique programming entity that has *methods* has *attributes* and can react to events.
Method - Things which an object can do; the "verbs" of objects. In code, usually can be identified by an "action" word.
Attribute- Things which describe an object; the "adjectives" of objects. In code, usually can be identified by a "descriptive" word- *enable, BackColor*.
Events- Forces external to an object to which that object can react. In code, usually attached to an event procedure.

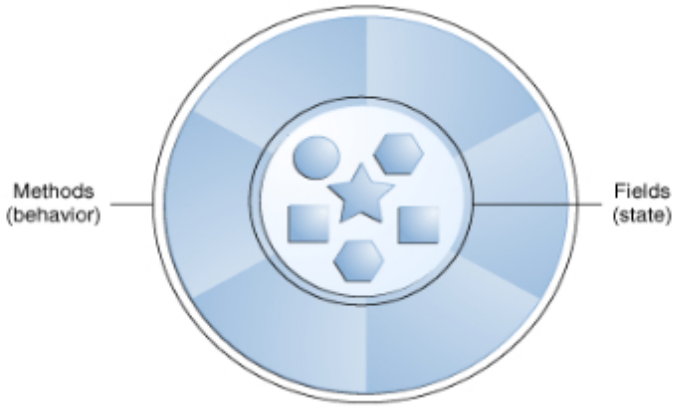


Figure 3: A software object

Object Oriented Programming Concepts III

► Software objects are conceptually similar to real-world objects: they too consist of state and related behavior.
► An object stores its state in fields (variables in some programming languages) and exposes its behavior through methods (functions in some programming languages).
► Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.
► Hiding internal state and requiring all interaction to be performed through an object's methods is known as data encapsulation — a fundamental principle of object-oriented programming.

Object Oriented Programming Concepts IV

Consider a bicycle, for example:

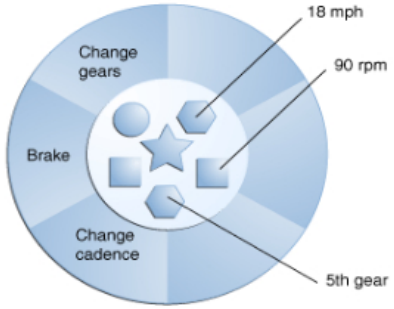


Figure 4: A bicycle modeled as a software object

Object Oriented Programming Concepts V

By attributing state (current speed, current pedal cadence, and current gear) and providing methods for changing that state, the object remains in control of how the outside world is allowed to use it. For example, if the bicycle only has 6 gears, a method to change gears could reject any value that is less than 1 or greater than 6.
Bundling code into individual software objects provides a number of benefits, including:

1. **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.
2. **Information-hiding:** By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.

Object Oriented Programming Concepts VI

3. **Code re-use:** If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.
4. **Pluggability and debugging ease:** If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace it, not the entire machine.

Class I

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components.

► In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles.

► A class is the blueprint from which individual objects are created.

The following Bicycle class is one possible implementation of a bicycle:

Class II

```
class Bicycle {
    int cadence = 0;
    int speed = 0;
    int gear = 1;
    void changeCadence(int newValue) {
   cadence = newValue;
    }
    void changeGear(int newValue) {
   gear = newValue;
    }
    void speedUp(int increment) {
   speed = speed + increment;
    }
```

Class III

```
    void applyBrakes(int decrement) {
   speed = speed - decrement;
    }
    void printStates() {
   System.out.println("cadence:" +
 cadence + " speed:" +
 speed + " gear:" + gear);
    }
}
```

Here's a BicycleDemo class that creates two separate Bicycle objects and invokes their methods:
class BicycleDemo

Class IV

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

// Create two different
// Bicycle objects
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();

// Invoke methods on
// those objects
bike1.changeCadence(50);
bike1.speedUp(10);
bike1.changeGear(2);
bike1.printStates();
```

```
  bike2.changeCadence(50);
  bike2.speedUp(10);
  bike2.changeGear(2);
  bike2.changeCadence(40);
  bike2.speedUp(10);
  bike2.changeGear(3);
  bike2.printStates();
    }
}
```

The output of this test prints the ending pedal cadence, speed, and gear for the two bicycles:

Class VI

```
cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3
```

Inheritance I

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

▶ The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

▶ Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

▶ In object-oriented programming, the concept of IS-A is a totally based on Inheritance.

▶ It is just like saying "A is a B type of thing". For example, Apple is a Fruit, Car is a Vehicle etc.

▶ Inheritance is uni-directional. For example, House is a Building. But Building is not a House.

Inheritance II

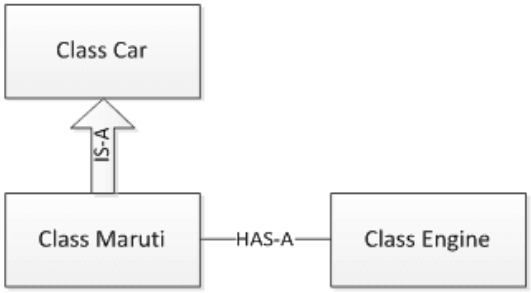


Figure 5: IS-A relationship

Why use inheritance in Java:

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Inheritance III

Terms used in Inheritance:

1. Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
2. Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
3. Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
4. Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Inheritance IV

▶ Different kinds of objects often have a certain amount in common with each other. For example: Mountain bikes, road bikes, and tandem bikes, all share the characteristics of bicycles (current speed, current pedal cadence, current gear).
▶ Yet each also defines additional features that make them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.
▶ Object-oriented programming allows classes to inherit commonly used state and behavior from other classes. In this example, Bicycle now becomes the superclass of $MountainBike$, $RoadBike$, and $TandemBike$.
▶ In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*.

Inheritance V

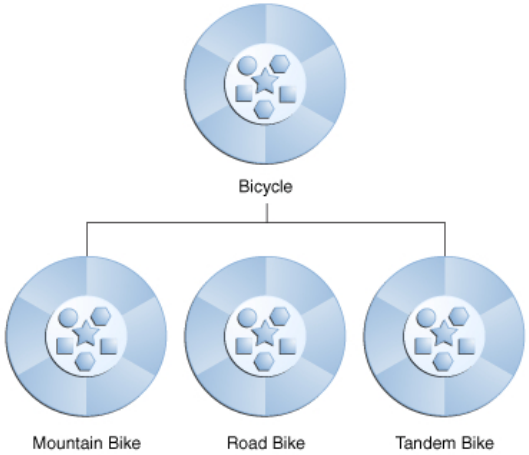


Figure 6: A hierarchy of bicycle classes

Inheritance VI

The syntax for creating a subclass is simple. At the beginning of your class declaration, use the extends keyword, followed by the name of the class to inherit from:

```
class MountainBike extends Bicycle {

    // new fields and methods defining
    // a mountain bike would go here

}
```

Inheritance VII

This gives MountainBike all the same fields and methods as Bicycle, **yet allows its code to focus exclusively on the features that make it unique**. This makes code for your subclasses easy to read.

▶ However, you must take care to properly document the state and behavior that each superclass defines, since that code will not appear in the source file of each subclass.

```
public class Bicycle {
    // the Bicycle class has three fields
    public int cadence;
    public int gear;
    public int speed;
    // the Bicycle class has one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
  gear = startGear;
```

Inheritance VIII

```
cadence = startCadence;
speed = startSpeed;
  }
  // the Bicycle class has four methods
  public void setCadence(int newValue) {
cadence = newValue;
  }
  public void setGear(int newValue) {
gear = newValue;
  }
  public void applyBrake(int decrement) {
speed -= decrement;
  }
  public void speedUp(int increment) {
```

Inheritance IX

```
  speed += increment;
    }
}
```

A class declaration for a MountainBike class that is a subclass of Bicycle might look like this:

```
public class MountainBike extends Bicycle {
    // the MountainBike subclass adds one field
    public int seatHeight;
    // the MountainBike subclass adds one method
    public void setHeight(int newValue) {
  seatHeight = newValue;
    }
}
```

Inheritance X

► MountainBike inherits all the fields and methods of Bicycle and adds the field seatHeight and a method to set it.

Another example:

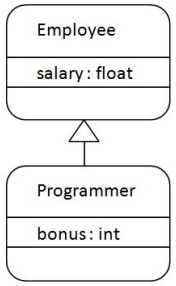


Figure 7: Inheritance

Inheritance XI

```
class Employee{
float salary=40000;
}
class Programmer extends Employee{
int bonus=10000;
public static void main(String args[]){
Programmer p=new Programmer();
Programmer p1=new Programmer();
p1.bonus=12000;
System.out.println("Programmer salary is:"+p.salary);
System.out.println("Bonus of Programmer is:"+p.bonus);

System.out.println("Programmer salary is:"+p1.salary);
```

Inheritance XII

```
System.out.println("Bonus of Programmer is:"+p1.bonus);

}
}
```

```
Output: ????
```

Inheritance XIII

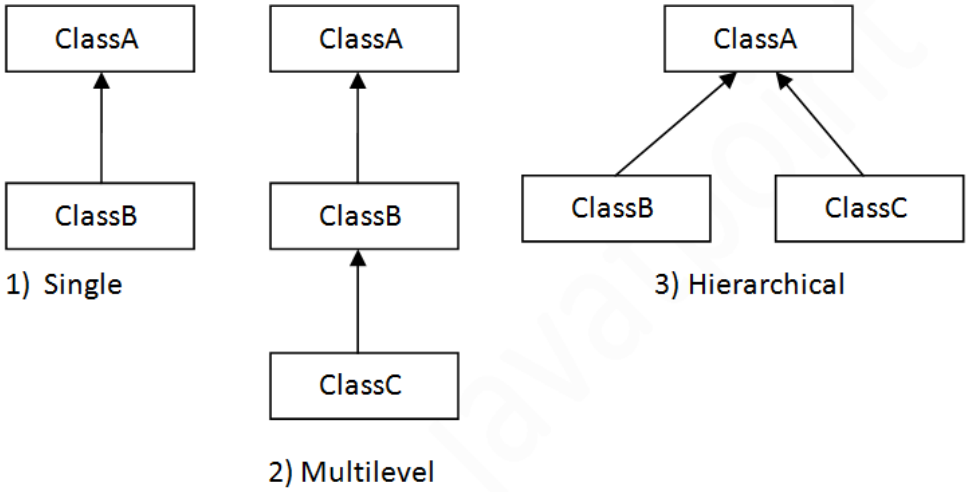


Figure 8: Types of inheritance in Java

Inheritance

Q) Why multiple inheritance is not supported in Java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in Java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

Inheritance

Q) Why multiple inheritance is not supported in Java?

► To reduce the complexity and simplify the language, multiple inheritance is not supported in Java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

Inheritance

Q) Why multiple inheritance is not supported in Java?

► To reduce the complexity and simplify the language, multiple inheritance is not supported in Java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

Inheritance

Q) Why multiple inheritance is not supported in Java?

► To reduce the complexity and simplify the language, multiple inheritance is not supported in Java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were
 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Polymorphism I

Polymorphism is considered one of the important features of Object-Oriented Programming.

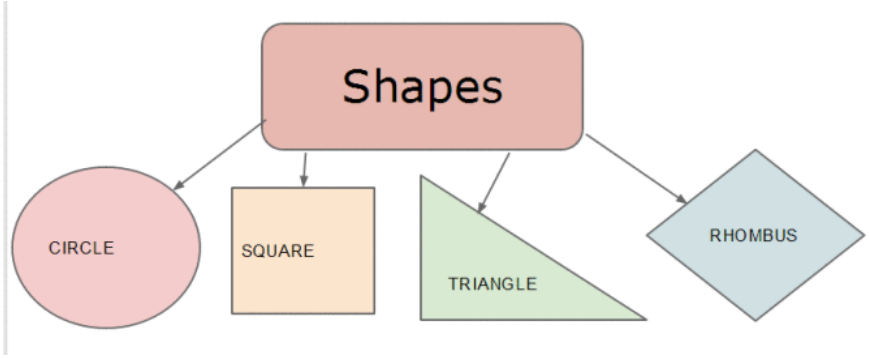
► Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.

► The word “poly” means many and “morphs” means forms, So it means many forms.

More generally, the concept of polymorphism is often expressed by the phrase “one interface, multiple methods.”

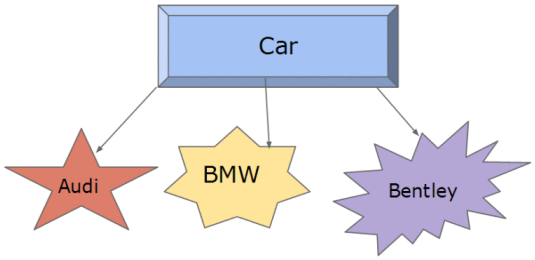
Polymorphism II

For example, Shapes is a class, and square, circles are all shapes which they can acquire. So a method written in shapes can be used by its forms like this:



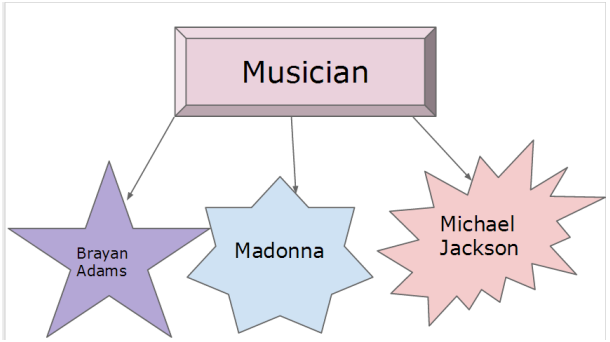
Polymorphism III

Cars can be of different types- Racing cars, 2 seater cars, 6 seater cars, 4 seater cars. Also can be classified on the basis of doors like cars having 2 doors, 4 doors, 6 doors, etc. It can also be categorized on the basis of brands like BMW, Audi, Merc, Maruti, etc.



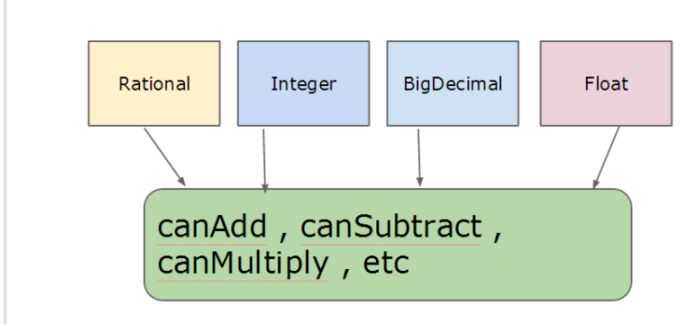
Musician – Different forms of musicians are there who sings Hollywood songs, Metal, Blues, Soft music, Rock Music, etc. So we can divide on the basis of singers-genres.

Polymorphism IV



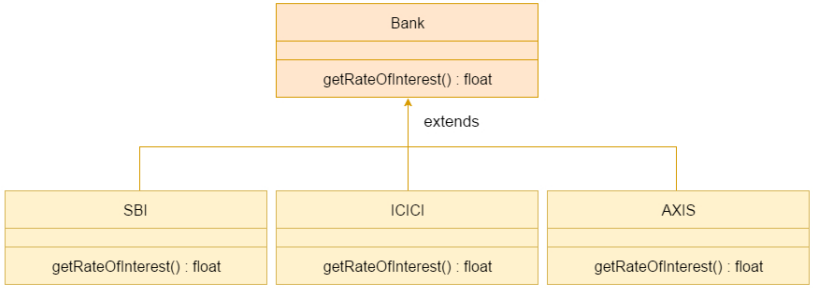
Numbers- Mathematics numbers are of different types like Rational, Integer, Big decimal, Float, etc. And they further can be used to apply other mathematical operations.

Polymorphism V



Consider a scenario where Bank is a class that provides a method to get the rate of interest. However, the rate of interest may differ according to banks.
For example, SBI, ICICI, and AXIS banks are providing 8.4%, 7.3%, and 9.7% rate of interest.

Polymorphism VI



Another Real life example of polymorphism: A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.

Polymorphism VII

Mr. Jhon

Mr. Jhon behave like **'Son'** with his parents

Mr. Jhon behave like **'Husband'** with his wife

Mr. Jhon behave like **'Father'** with his daughter

Mr. Jhon behave like **'Employee'** in Office

Polymorphism VIII

Polymorphism example in the context of Java:

```
// Java program for Method overloading
class MultiplyFun {
    // Method with 2 parameter
    static int Multiply(int a, int b){
        return a * b;
    }
    // Method with the same name but 3 parameter
    static int Multiply(int a, int b, int c){
        return a * b * c;
    }
}
class Main {
```

Polymorphism IX

```
    public static void main(String[] args)
    {
        System.out.println(MultiplyFun.Multiply(2, 4));
   System.out.println(MultiplyFun.Multiply(2, 7, 3));
    }
}
```

Polymorphism X

Advantage of polymorphism:

- ✓ It helps the programmer to reuse the codes, i.e., classes once written, tested and implemented can be reused as required.
- ✓ Saves a lot of time.
- ✓ Single variable can be used to store multiple data types.
- ✓ Easy to debug the codes.

Data Abstraction I

► Abstraction is a process of hiding the implementation details and showing only functionality to the user.
► Another way, it shows only essential things to the user and hides the internal details.
► For example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Data Abstraction II

A car is viewed as a car rather than its individual components.

Registration

- Vehicle Identification Number
- License plate
- Current Owner
- Tax due, date

Owner

- Car Description
- Service History
- Petrol Mileage History

Garage

- License plate
- Work Description
- Billing Info
- Owner

Data Abstraction III

► In the figure, you can see that an Owner is interested in details like Car description, service history, etc.
► Garage Personnel are interested in details like License, work description, bill, owner, etc.
► Registration Office interested in details like vehicle identification number, current owner, license plate, etc. It means each application identifies the details that are important to it.
► Abstraction can be seen as the technique of filtering out the unnecessary details of an object so that there remain only the useful characteristics that define it. Abstraction focuses on the perceived behavior of the entity. It provides an external view of the entity.
► Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Data Abstraction IV

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

► **Abstract classes and Abstract methods :**

1. An abstract class is a class that is declared with abstract keyword.
2. An abstract method is a method that is declared without an implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods.

Data Abstraction V

4. A method defined abstract must always be redefined in the subclass,thus making overriding compulsory OR either make subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with abstract keyword.
6. There can be no object of an abstract class.That is, an abstract class can not be directly instantiated with the new operator.
7. An abstract class can have parametrized constructors and default constructor is always present in an abstract class.

► When to use abstract classes and abstract methods

There are situations in which we will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. That is, sometimes we will want to create a superclass that only defines a generalization form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.

Data Abstraction VI

Consider a classic “shape” example, perhaps used in a computer-aided design system or game simulation. The base type is “shape” and each shape has a color, size and so on. From this, specific types of shapes are derived(inherited)-circle, square, triangle and so on – each of which may have additional characteristics and behaviors. For example, certain shapes can be flipped. Some behaviors may be different, such as when you want to calculate the area of a shape. The type hierarchy embodies both the similarities and differences between the shapes.

Data Abstraction VII

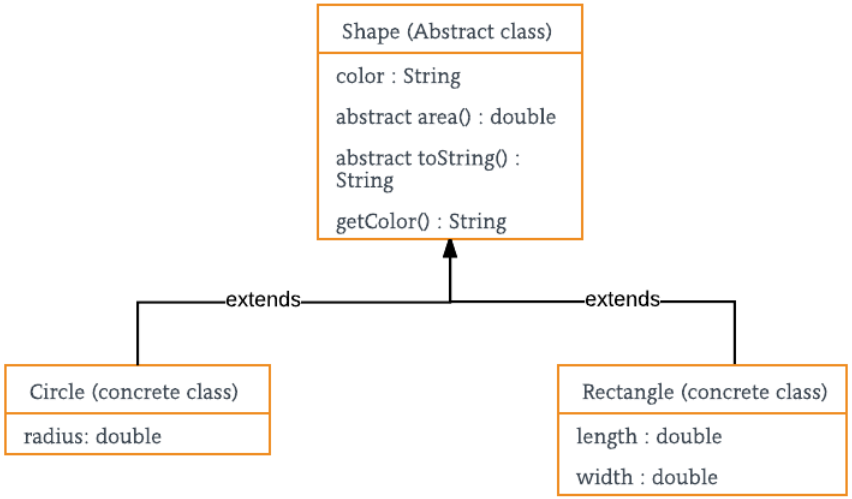
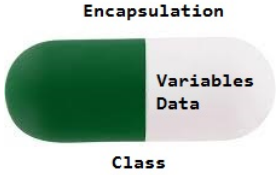


Figure 9: Abstract Class and Methods

Encapsulation I

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates.

► Other way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.



Encapsulation II

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of class as private and the class is exposed to the end user or the world without providing any details behind implementation using the abstraction concept, so it is also known as combination of data-hiding and abstraction.
- Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

Advantages of Encapsulation:

Encapsulation III

- Data Hiding: The user will have no idea about the inner implementation of the class. It will not be visible to the user that how the class is storing values in the variables. He only knows that we are passing the values to a setter method and variables are getting initialized with that value.
- Increased Flexibility: We can make the variables of the class as read-only or write-only depending on our requirement. If we wish to make the variables as read-only then we have to omit the setter methods. If we wish to make the variables as write-only then we have to omit the get methods.
- Reusability: Encapsulation also improves the re-usability and easy to change with new requirements. Testing code is easy: Encapsulated code is easy to test for unit testing.

Encapsulation IV

Example of encapsulation:

```
class Employee{
   private int Emp_id; //Data hiding
   public void SetEmpId(int Emp_id1){
      Emp_id = Emp_id1;
   }
   public int GetEmpId(){
      return Emp_id;
   }
}
```

Encapsulation V

```
/* File name : EncapTest.java */
public class EncapTest {
   private String name;
   private String idNum;
   private int age;
   public int getAge() {
      return age;
   }
   public String getName() {
      return name;
   }
   public String getIdNum() {
      return idNum;
```

Encapsulation VI

```
   }
   public void setAge( int newAge) {
      age = newAge;
   }
   public void setName(String newName) {
      name = newName;
   }
   public void setIdNum( String newId) {
      idNum = newId;
   }
}
/* File name : RunEncap.java */
public class RunEncap {
```

Encapsulation VII

```
   public static void main(String args[]) {
      EncapTest encap = new EncapTest();
      encap.setName("James");
      encap.setAge(20);
      encap.setIdNum("12343ms");
      System.out.print("Name : " + encap.getName() + " Age : " +
          encap.getAge());
   }
}
```

```
Name : James Age : 20
```

Encapsulation vs Data Abstraction I

S.NO	ABSTRACTION	ENCAPSULATION
1.	Abstraction is the process or method of gaining the information.	While encapsulation is the process or method to contain the information.
2.	In abstraction, problems are solved at the design or interface level.	While in encapsulation, problems are solved at the implementation level.
3.	Abstraction is the method of hiding the unwanted information.	Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.

Encapsulation vs Data Abstraction II

4.	We can implement abstraction using abstract class and interfaces.	Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public.
5.	In abstraction, implementation complexities are hidden using abstract classes and interfaces.	While in encapsulation, the data is hidden using methods of getters and setters.
6.	The objects that help to perform abstraction are encapsulated.	Whereas the objects that result in encapsulation need not be abstracted.

Encapsulation vs Data Abstraction III

Polymorphism, Encapsulation, and Inheritance Work Together I

► When properly applied, polymorphism, encapsulation, and inheritance combine to produce a programming environment that supports the development of far more robust and scaleable programs than does the process-oriented model.
► A well-designed hierarchy of classes is the basis for reusing the code in which you have invested time and effort developing and testing.
► Encapsulation allows you to migrate your implementations over time without breaking the code that depends on the public interface of your classes.
► Polymorphism allows you to create clean, sensible, readable, and resilient code.

Polymorphism, Encapsulation, and Inheritance Work Together II

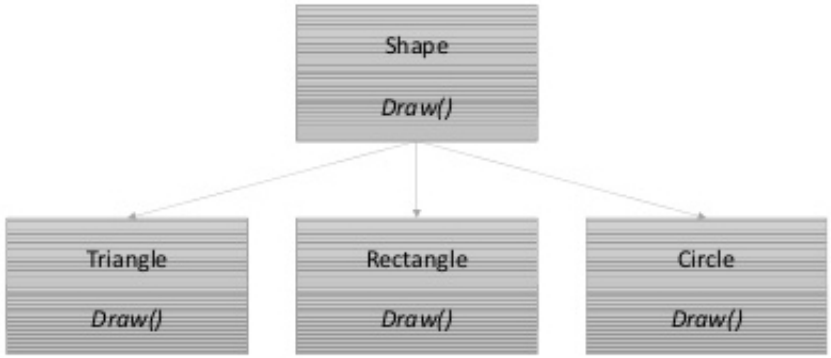


Figure 10: Polymorphism, Encapsulation, and Inheritance work together

Polymorphism, Encapsulation, and Inheritance Work Together III