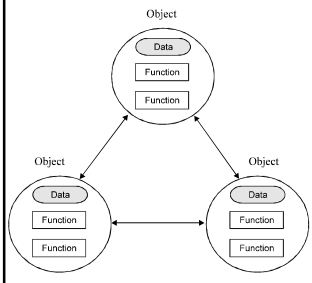
Introduction to Object Oriented Design

Overview

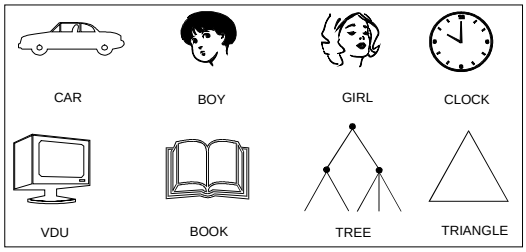
- Understand Classes and Objects.
- Understand some of the key concepts/features in the Object Oriented paradigm.
- Benefits of Object Oriented Design paradigm.

OOP: model, map, reuse, extend

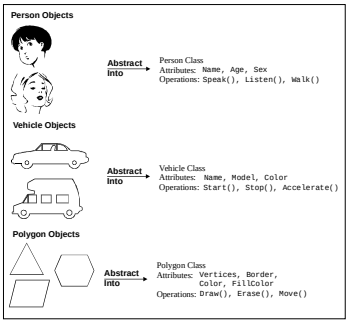


- Model the real world problem to user's perceive;
- Use similar metaphor in computational env.
- Construct reusable components;
- Create new components from existing ones.

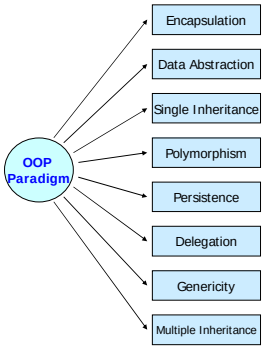
Examples of Objects



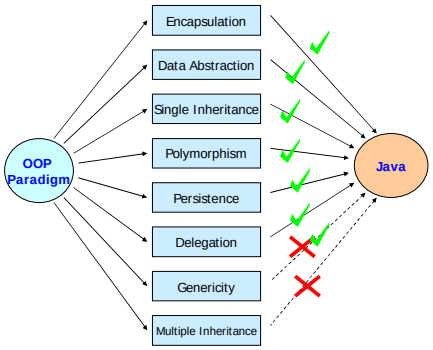
Classes: Objects with the same attributes and behavior



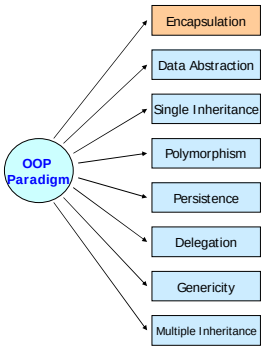
Object Oriented Paradigm: Features



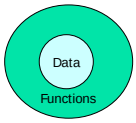
Java's OO Features



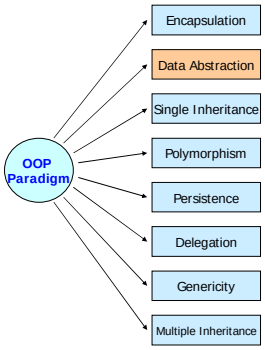
Encapsulation



- It associates the code and the data it manipulates into a single unit; and keeps them safe from external interference and misuse.



Data Abstraction



- The technique of creating new data types that are well suited to an application.
- It allows the creation of user defined data types, having the properties of built data types and a set of permitted operators.
- In Java, partial support.
- In C++, fully supported (e.g., operator overloading).

Abstract Data Type (ADT)

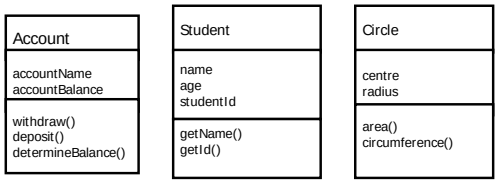
- A structure that contains both data and the actions to be performed on that data.
- *Class* is an implementation of an Abstract Data Type.

Class- Example

```
class Account {
   private String accountName;
   private double accountBalance;

    public  withdraw();
   public  deposit();
   public  determineBalance();
} // Class Account
```


Class

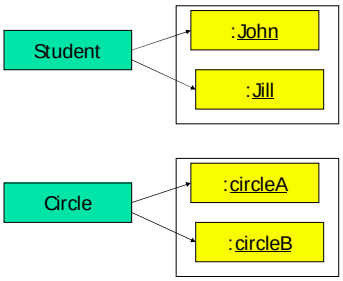
- Class is a set of *attributes* and *operations* that are performed on the attributes.



Objects

- An Object Oriented system is a collection of interacting Objects.
- Object is an instance of a class.

Classes/Objects



Class

- A class represents a template for several objects that have common properties.
- A class defines all the properties common to the object - *attributes* and *methods*.
- A class is sometimes called the object's type.

Object

- Objects have state and classes don't.

John is an object (instance) of class Student.
name = "John", age = 20, studentId = 1236

Jill is an object (instance) of class Student.
name = "Jill", age = 22, studentId = 2345

circleA is an object (instance) of class Circle.
centre = (20,10), radius = 25

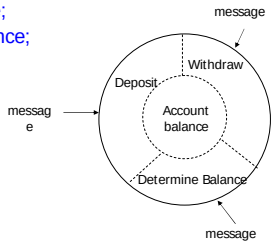
circleB is an object (instance) of class Circle.
centre = (0,0), radius = 10

Encapsulation

- All information (attributes and methods) in an object oriented system are stored within the object/class.
- Information can be manipulated through operations performed on the object/class – interface to the class. Implementation is hidden from the user.
- Object support *Information Hiding* – Some attributes and methods can be hidden from the user.

Encapsulation - Example

```
class Account {
   private String accountName;
   private double accountBalance;

   public  withdraw();
   public  deposit();
   public  determineBalance();
} // Class Account
```



Data Abstraction

- The technique of creating new data types that are well suited to an application.
- It allows the creation of user defined data types, having the properties of built in data types and more.

Abstraction - Example

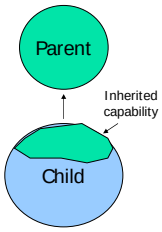
```
class Account {
    private String accountName;
    private double accountBalance;

    public  withdraw();
    public  deposit();
    public  determineBalance();
} // Class Account
```

Creates a data type Account

Account acctX;

Inheritance

- New data types (classes) can be defined as extensions to previously defined types.
- Parent Class (Super Class) – Child Class (Sub Class)
- Subclass inherits properties from the parent class.

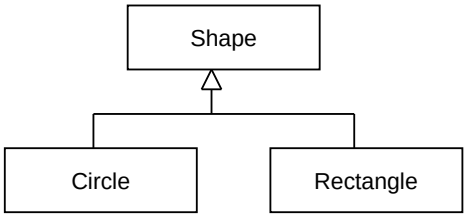


Inheritance - Example

- Example
  - Define Person to be a *class*
    - A Person has *attributes*, such as age, height, gender
    - Assign values to attributes when describing object
  - Define student to be a *subclass* of Person
    - A student has all attributes of Person, plus attributes of his/her own (student no, course_enrolled)
    - A student has all attributes of Person, plus attributes of his/her own (student no, course_enrolled)
    - A student *inherits* all attributes of Person
  - Define lecturer to be a *subclass* of Person
    - Lecturer has all attributes of Person, plus attributes of his/her own (staff_id, subjectID1, subjectID2)

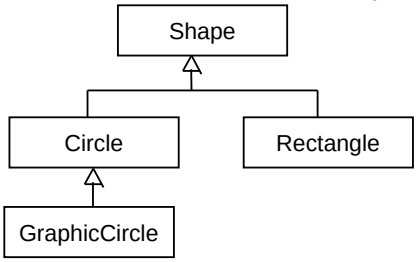
Inheritance - Example

- Circle Class can be a subclass (inherited from) of a parent class - Shape



Inheritance - Example

- Inheritance can also have multiple levels.

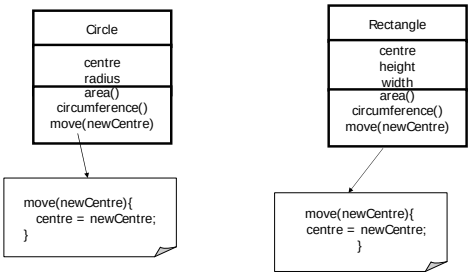


Uses of Inheritance - Reuse

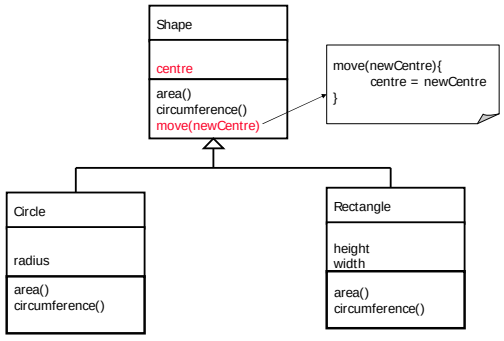
- If multiple classes have common attributes/methods, these methods can be moved to a common class - parent class.
- This allows reuse since the implementation is not repeated.

Example : Rectangle and Circle method have a common method move(), which requires changing the centre coordinate.

Uses of Inheritance - Reuse



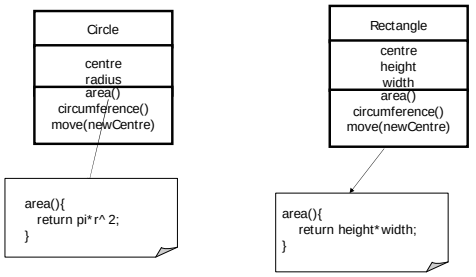
Uses of Inheritance - Reuse



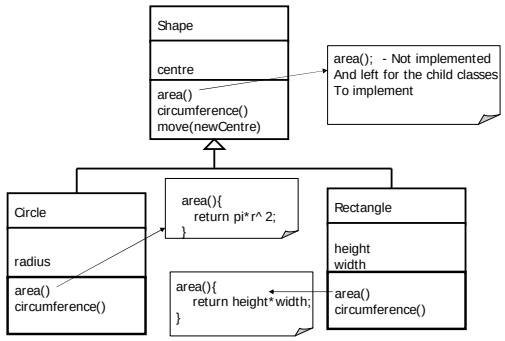
Uses of Inheritance - Specialization

- Specialized behavior can be added to the child class.
- In this case the behaviour will be implemented in the child class.
  - E.g. The implementation of area() method in the Circle class is different from the Rectangle class.
- area() method in the child classes override the method in parent classes().

Uses of Inheritance - Specialization



Uses of Inheritance - Specialization



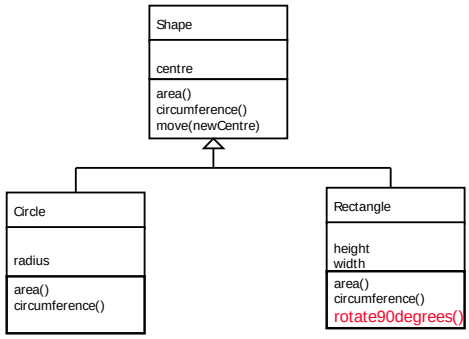
Uses of Inheritance – Common Interface

- All the operations that are supported for Rectangle and Circle are the same.
- Some methods have common implementation and others don't.
  - move() operation is common to classes and can be implemented in parent.
  - circumference(), area() operations are significantly different and have to be implemented in the respective classes.
- The Shape class provides a common interface where all 3 operations *move(), circumference()* and *area()*.

Uses of Inheritance - Extension

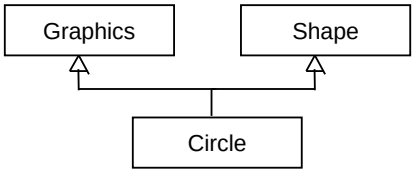
- Extend functionality of a class.
- Child class adds new operations to the parent class but does not change the inherited behavior.
  - E.g. Rectangle class might have a special operation that may not be meaningful to the Circle class - rotate90degrees()

Uses of Inheritance - Extension



Uses of Inheritance – Multiple Inheritance

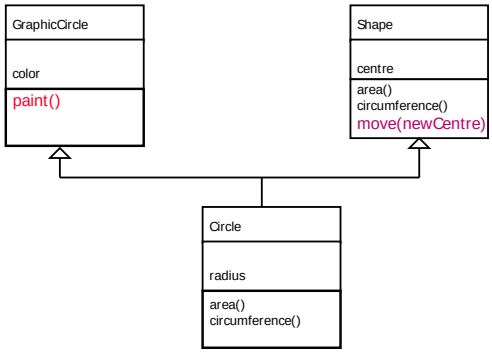
- Inherit properties from more than one class.
- This is called *Multiple Inheritance*.



Uses of Multiple Inheritance

- This is required when a class has to inherit behavior from multiple classes.
- In the example Circle class can inherit *move()* operation from the Shape class and the *paint()* operation from the Graphics class.
- Multiple inheritance is not supported in JAVA but is supported in C++.

Uses of Inheritance – Multiple Inheritance



Polymorphism

- Polymorphic which means “many forms” has Greek roots.
  - Poly – many
  - Morphos - forms.
- In OO paradigm polymorphism has many forms.
- Allow a single *object, method, operator* associated with different meaning depending on the type of data passed to it.

Polymorphism

- An object of type Circle or Rectangle can be assigned to a Shape object. The behavior of the object will depend on the object passed.

```
circleA = new Circle();        Create a new circle object

Shape shape = circleA;
shape.area();   area() method for circle class will be executed


rectangleA = new Rectangle(); Create a new rectangle object
shape= rectangle;
shape.area()     area() method for rectangle will be executed.
```


Polymorphism – Method Overloading

- Multiple methods can be defined with the same name, different input arguments.
  Method 1 - initialize(int a)
  Method 2 - initialize(int a, int b)
- Appropriate method will be called based on the input arguments.
  initialize(2) Method 1 will be called.
  initialize(2,4) Method 2 will be called.

Polymorphism – Operator Overloading

- Allows regular operators such as +, -, *, / to have different meanings based on the type.
- E.g. + operator for Circle can re-defined
  Circle c = c + 2;
- Not supported in JAVA. C++ supports it.

Persistence

- The phenomenon where the object outlives the program execution.
- Databases support this feature.
- In Java, this can be supported if users explicitly build object persistency using IO streams.

Why OOP?

- Greater Reliability
  - Break complex software projects into small, self-contained, and modular objects
- Maintainability
  - Modular objects make locating bugs easier, with less impact on the overall project
- Greater Productivity through Reuse!
- Faster Design and Modelling

Benefits of OOP..

- Inheritance: Elimination of Redundant Code and extend the use of existing classes.
- Build programs from existing working modules, rather than having to start from scratch. → save development time and get higher productivity.
- Encapsulation: Helps in building secure programs.

Benefits of OOP..

- Multiple objects to coexist without any interference.
- Easy to map objects in problem domain to those objects in the program.
- It is easy to partition the work in a project based on objects.
- The Data-Centered Design enables us in capturing more details of model in an implementable form.

Benefits of OOP..

- Object Oriented Systems can be easily upgraded from small to large systems.
- Message-Passing technique for communication between objects make the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

Summary

- Object Oriented Design, Analysis, and Programming is a Powerful paradigm
- Enables Easy Mapping of Real world Objects to Objects in the Program
- This is enabled by OO features:
  - Encapsulation
  - Data Abstraction
  - Inheritance
  - Polymorphism
  - Persistence
- Standard OO Design (UML) and Programming Languages (C++/Java) are readily accessible.

Reference

- Chapter 1: “Programming with Java” by Balagurusamny
- Optional:
  - Chapter 1: “Mastering C++” by V. Rajuk and R. Buyya, Tata McGraw Hill, New Delhi, India.