Object Oriented Programming

Object Oriented Programming with Java - Chapter 3

FPTU Da Nang – IT Department

Objectives - Content

- Think OOP
- Object concept – Class
- 3 features of OOP
- Structure of class
  - Attributes
  - Methods
  - Construction
- Data abstraction
- Encapsulation
- Inheritance
- Polymorphism

Think OOP

OOP concept

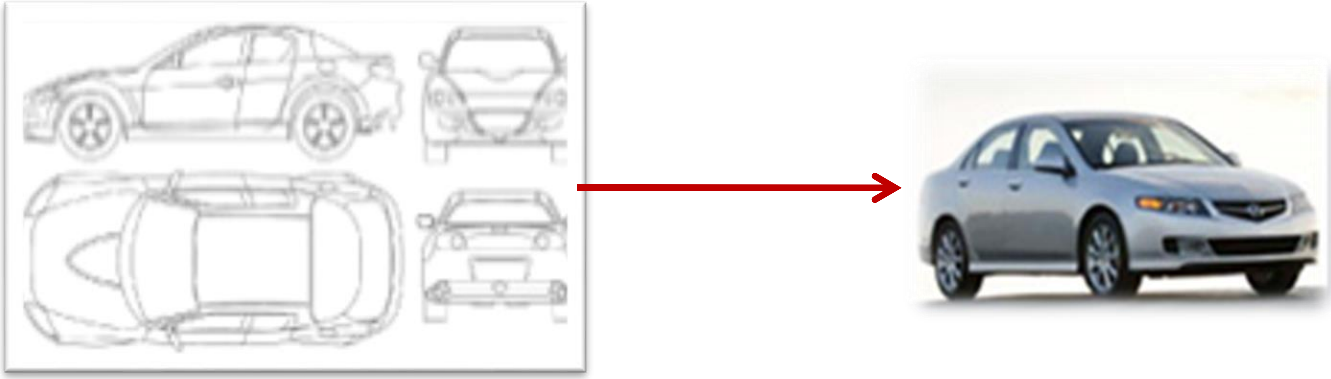
- Starting point: All are objects.
  - The real world are objects
  - Each object has properties and behaviors
  - Objects interact with each other.
- New ways of thinking to solve problems on computers
  - adapt the computer to the problem - instead of describing the problem according to what is familiar to computer.
  - Replace data structures and data manipulation functions with objects
  - Create a new type of association - stronger between data and data processing functions
  - New way of organizing source code

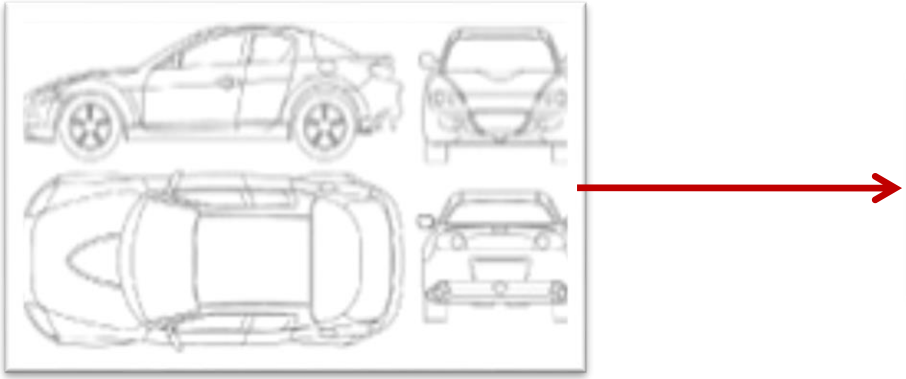
OOP concept..

- **OOP programming is**
  - indicate the objects that relevant the problem,
  - simulate objects with classes,
  - let objects to interact with each other.
- OOP techniques help describe real-world problems into relationships between objects in a programming problem.
  - Tightly associate real-world objects with programming objects
  - OOP programs are images of objects exchanging information (sending messages).
  - Way for programmers to describe the problem, solving the problem
- Writing a sales management program, how would you start?

Advantages of OOP

- OOP techniques guide the process of analyzing, designing & implementing software projects, using project business concepts & terminology.
  - expressed in a business language instead of a programming language
  - Object-oriented analysis and design
- Increase programming productivity & efficiency
  - **Reusable code**
  - **Ability to edit, upgrade, customize quickly.**
  - **Promote sharing of programming source codes**
- Ease of isolation & error handling
- Limit accidental errors caused by random data modification





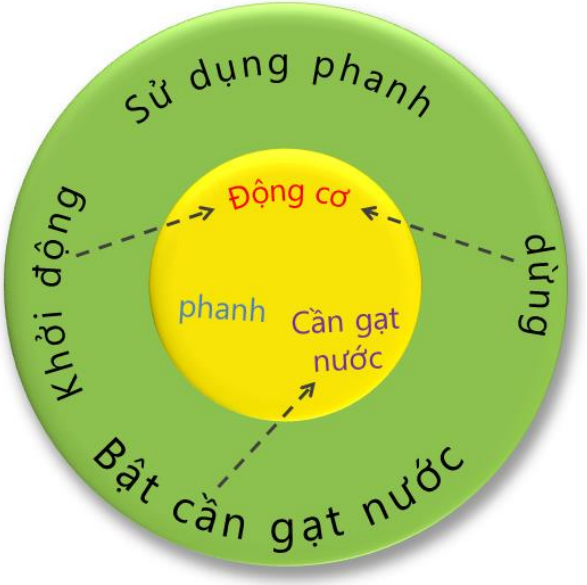
Class

Object

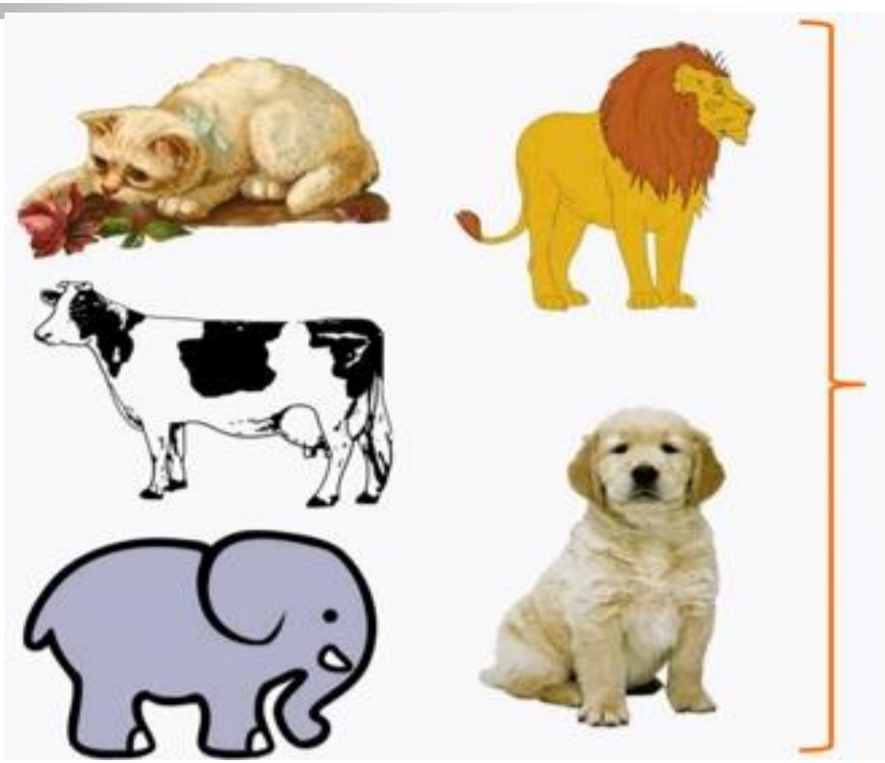
Properties

  - Manufacturer
  - Model
  - Year
  - Color

- Behavior
  - Start up
  - Stop
  - Brake
  - Turn on the wipers



Group of Cars



Group of Animals

Object - Class

- The components involved in a problem are represented as objects.
  - The object is a black box that hides its internal structure, communicating with the outside by sending & receiving messages.
  - An object is actually a collection of data and operations on that data set.
- An object includes:
  - Attributes: data, information about the object
  - Behaviors: methods, operations that change the properties of the object
- Objects of the same type are described as class

Class

- A class is a description of a collection of objects that share the same properties (data) and behavior (method).
- Class is actually a data type
  - defined in accordance with the problem
  - not just a unit of storage on a computer.
- The class is built from the following components:
  - Component data – data member (attribute)
  - Member function – function member (behavior)
- Ex: class Fraction: represent fraction
  - numerator, denominator
  - Add, subtract, multiply, divide, inverse, simplify fraction

Relationship between class and object

- class defines an entity - while object is that entity
- class is the template - conceptual model - for objects.
  - All objects of a class share the same characteristics & actions.
- General – particular instance
- General – specific thing

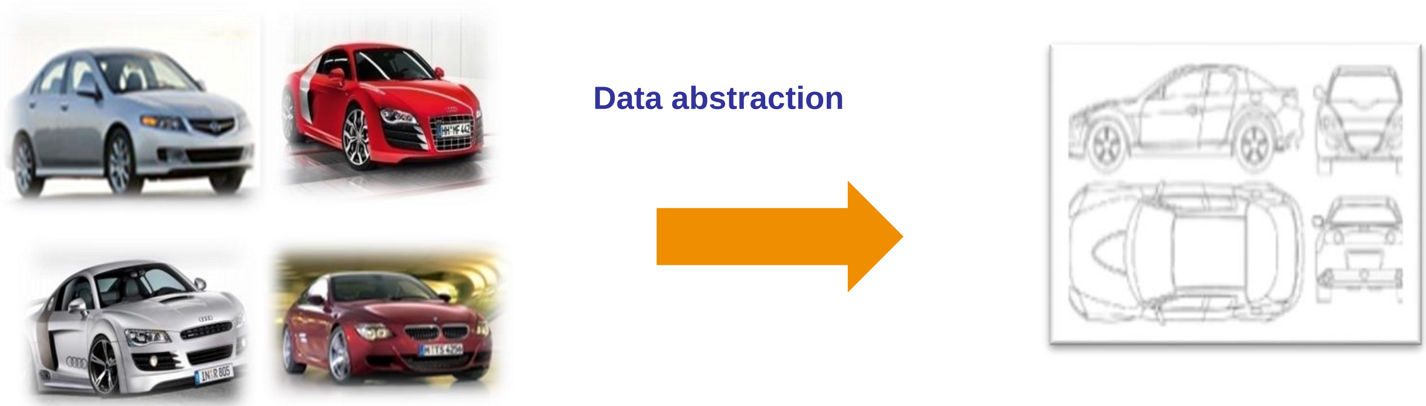
Object attributes

- Attribute, property, field, data member : nouns
  - Use nouns to name attribute.
- Store the data of an object.
  - The value of the data is determined only after the object is created
  - Are member variables declared inside a class
- Shared attributes between all objects
  - “School name” of all students in a school
  - Static variable, class variable: use static keyword
- Instance attributes, individual attributes
  - Student's "Date of Birth"
  - Each object has it own attributes

Object behaviors

- Behavior, method, function member : verbs
- Methods, functions, behaviors:
  - use verbs to name method
- A method is an object's behavior
  - In programming terms, a method is a function
  - Defines the steps of an action to be performed by the object when the method invocation is passed to the object.
  - A way to change the status - instance variables- of an object.
- Method define the interface of an object
  - "send a message to the object"
  - Ways for objects comunication

Data abstraction???



Data abstraction

Objects

Class

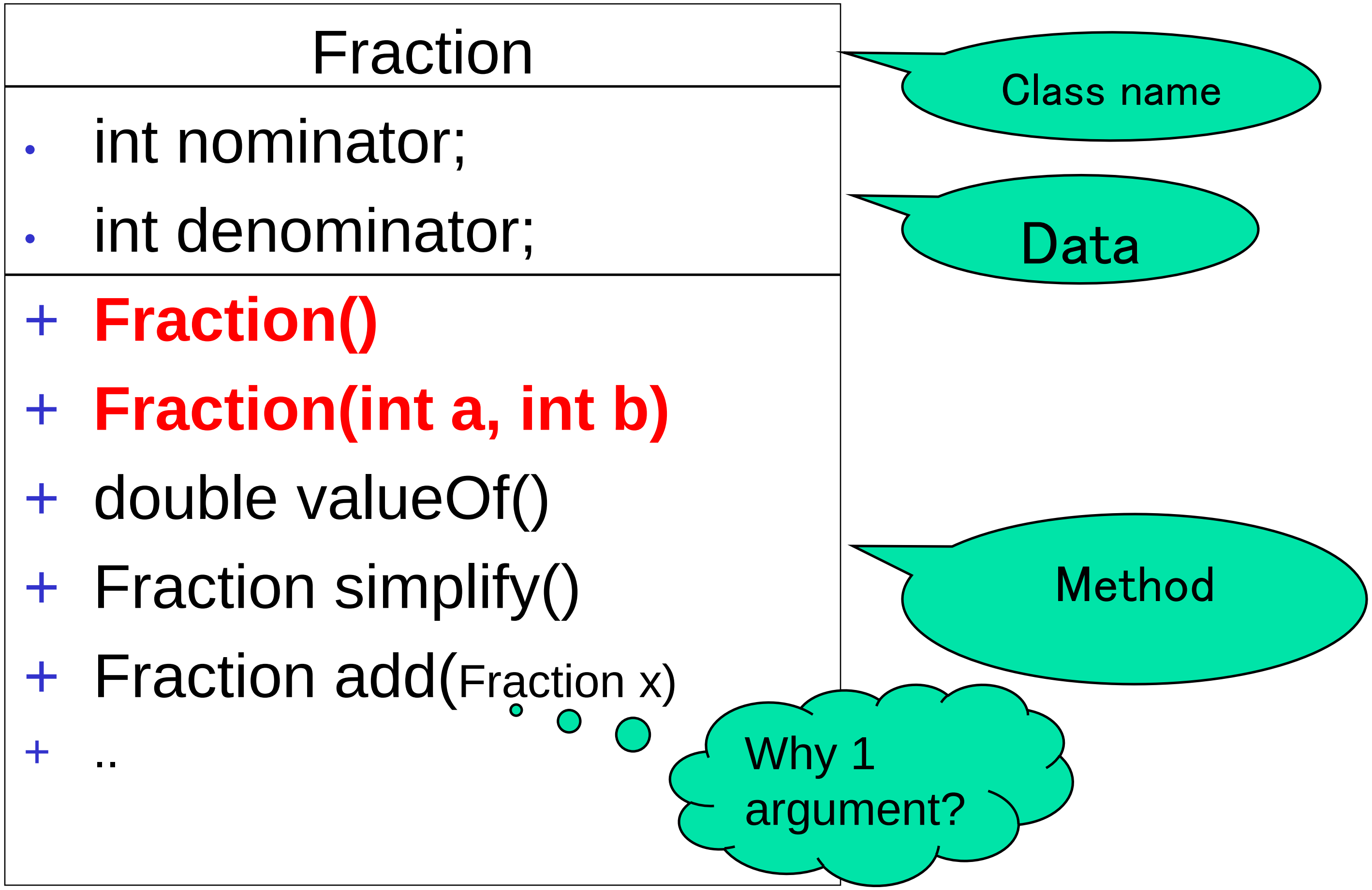
Data abstraction

- **Data abstraction**
  - The process of identifying objects with properties and behaviors that are appropriate with the problem being solved.
  - One of the challenges of OOP is to create a one-to-one mapping between the elements of the real problem and the objects in the programming problem.
  - Design class diagrams of objects
- Creating abstract data types (classes) - is the fundamental issue of OOP.
- The abstract model simplifies the real-world problem but must accurately reflect the real world in order to be able to use the model for predicting real-world behaviour.

Data abstraction..

- Advantages of data abstraction:
  - Identify and focus on the problem at hand
  - Get rid of uninteresting details
- Data abstraction method:
  3 questions to answer:
  - What are the objects?
    - How to decompose a problem into objects
  - What are their interfaces?
    - What kind of information, what messages will objects exchange?
  - What properties are concerned?
    - What kind of data are we interested in?

Class diagram



Fraction

- int nominator;
- int denominator;

+ **Fraction()**

+ **Fraction(int a, int b)**

+ double valueOf()

+ Fraction simplify()

+ Fraction add(Fraction x)

+ ..

Class name

Data

Method

Why 1 argument?

Class definition - syntax

-
  ```
  class ClassName {
   //declare data member: variables, properties
   //declare function member: function, behavior
  }
  ```

-
  ```
  class Fraction {
   private int nom, denom; // instance variable
   public Fraction(int t,m) { // constructor
     nom=t; denom=m;
   }
   public Fraction simplify() { //returns fraction after simplification
       // code to simplify a fraction
       return this;
   }
  }
  ```

Class definition

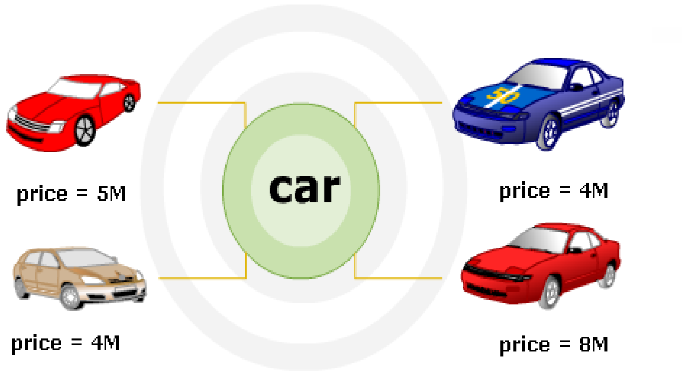
- Components of class
  - Data member : 2 types
    + instance variable – the unique properties of each object
    + static variable – common properties for all objects
  - Function members: 3 types
    + Constructor
    + Method - the behavior of the object
    + Static function – library function, independent with all objects
- class variables and functions declared after the static . keyword
  - Ex: schoolName is a common attribute for all students of a school – static var.
  - studentID, fullName, dateOfBirth are unique properties of each student – instance var.
  - class Student {

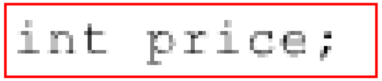
```
class Student {
 String studentID, fullName;
 Date dob;
 static String schoolName;
 //…
}
```

Instance variables

- Used to store information about an entity.

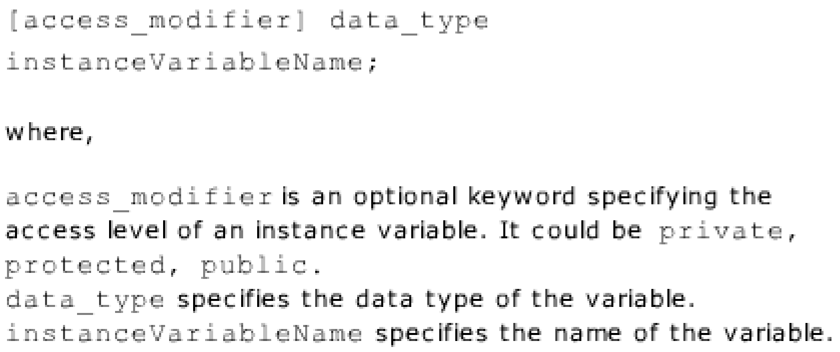
- Instance variables are declared in the same way as local variables.





```
int price;
```

Example



```
[access_modifier] data_type
instanceVariableName;
```

where,

`access_modifier` is an optional keyword specifying the access level of an instance variable. It could be `private`, `protected`, `public`.
`data_type` specifies the data type of the variable.
`instanceVariableName` specifies the name of the variable.

Constructor

- A special method for object initialization of a class.
  - allocate memory for data members
  - initialize the object's data members
  - Being called when instantiating object using ***new*** *operator*
    - *Fraction a= new Fraction(3,4);*
  - *All classes must have constructor*
- *Notes on syntax*
  - *Constructor name is the same as class name*
  - *No return type*
  - *The constructors call each other by syntax* ***this(argument);***
- *Default constructor*
  - *Has no arguments*
  - *automatically generated by the compiler when the class did not declare any constructor*
- *There can be multiple constructors up on the initialization context of an object*

Method - function

- **Method** - represent the behavior of object.
  - changes object's properties/ states
  - Can access all other members
  - Mechanism for interactions between objects
  - *Example: multiply method in class Fraction*

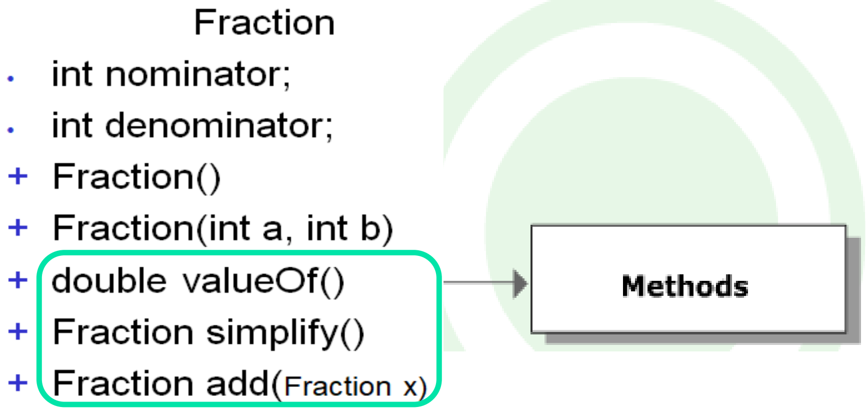
```
Fraction multiply(Fraction p) {
 return new Fraction(this.nom*p.nom, this.denom*p.denom);
}
```

- ***Static Function*** *– shared function, library function*
  - *Independent of class objects*
  - *static members can be accessed each others*
  - *Ex: static function to multiply 2 fractions*

```
static Fraction multiply(Fraction p, Fraction q){
    return p.multiply(q);}
```

Method

- A method is defined as the actual implementation of an operation on an object



Fraction

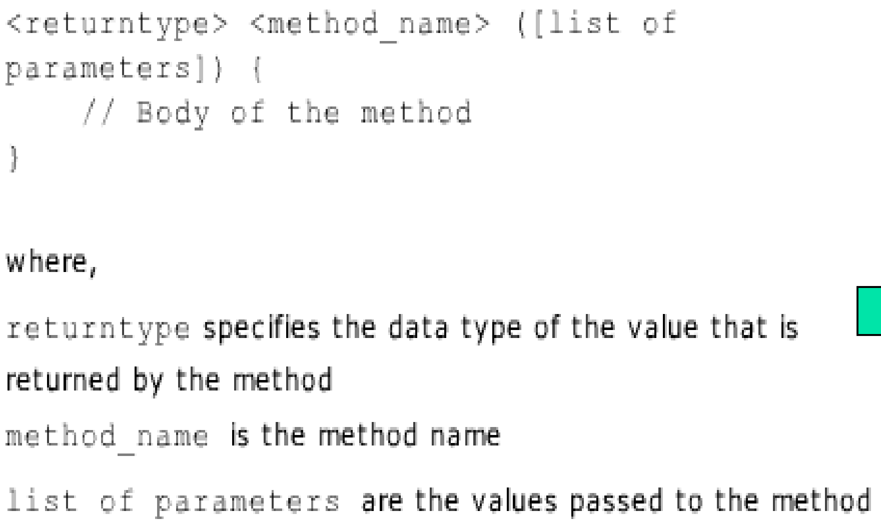
- int nominator;
- int denominator;
- + Fraction()
- + Fraction(int a, int b)
- + double valueOf()
- + Fraction simplify()
- + Fraction add(Fraction x)

- Syntax :

```
access_specifier modifier datatype method_name
    (parameter_list){
    //body of the method
}
```

Instance Method

- Invoked by an instance object and can access instance variables.



```
<returntype> <method_name> ([list of
parameters]) {
    // Body of the method
}
```

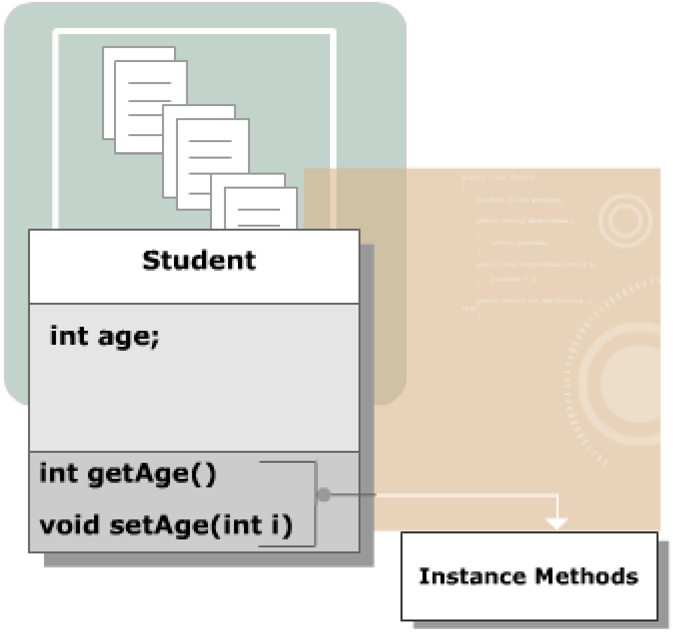
where,

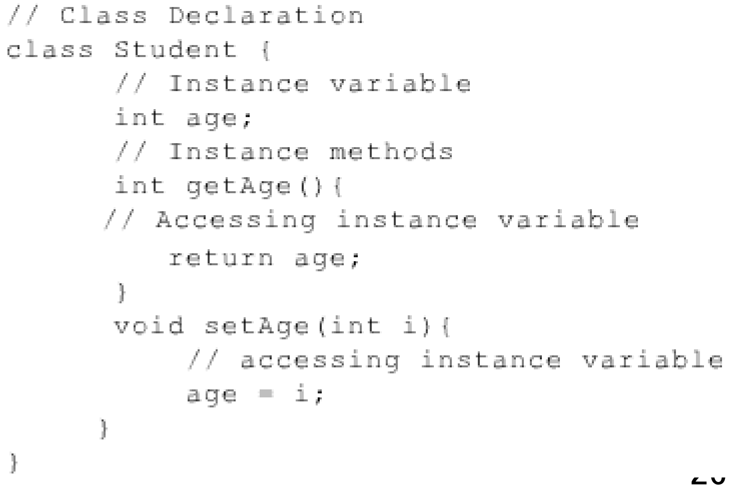
`returntype` specifies the data type of the value that is returned by the method

`method_name` is the method name

`list of parameters` are the values passed to the method

Ex.





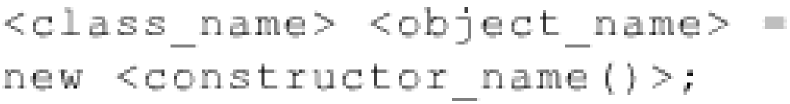
```
// Class Declaration
class Student {
     // Instance variable
     int age;
     // Instance methods
     int getAge(){
    // Accessing instance variable
        return age;
     }
     void setAge(int i){
         // accessing instance variable
         age = i;
       }
}
```

Access class members

- Access within the class
  - this.fieldName
  - this.methodName(argument)
  - *use* **this** *when it is necessary to distinguish member variables of a class has the same name as local variables.*
- Access from outside the class
  - Constructor: call the constructor using the **new** operator
  - Non-static member
    - VarName.fieldName
    - VarName.methodName(argument)
  - Static member
    - ClassName.memberName
- Static methods are only allowed to reference static members of the class

Creating object

- Objects are declared to represent the class.
- The **`new`** operator dynamically allocates memory for an object and returns a reference to it.
- All class objects must be dynamically allocated.



```
<class_name> <object_name> =
new <constructor_name()>;
```

Ex.

```
Fraction f1= new Fraction();
Fraction f2= new Fraction(numer, deno);
```

Three characteristics of OOP

- Encapsulation:
  - Encapsulation, data hiding
  - Protect data from unauthorized external access
  - Use access modifiers: **public, private, protected**
- Inherritance
  - Inheritance: the most "expensive" idea of OOP
  - Re-use the source code – code re-usable
  - Base Class (- parent class) and Derived Class (child)
  - Abstract class and interface
- Polymorphism
  - Diversity when subjects exhibit behaviors
  - Overload – override method

Encapsulation – data hiding

- Hide the internal structure of the class, including:
  - Properties
  - Method
  - Implementation details
- Access modifiers are used to implement the idea of encapsulation
  - public – private - protected
- Encapsulation
  - Data is always hide: member variables almost private access
  - Method for communication with the outside world: public access
  - Protected access for sub-class that inherited
- The encapsulation feature allows the internal content of the class to be changed without affecting the related classes

Encapsulation.. Keywords specifying access

- Access modifiers:
  - control access to class & class members
  - public
    - No limitation access
    - apply to class, applet, application, class members
  - private
    - only allow access inside the class
    - does not apply to class declaration
  - protected
    - allow access from subclass
  - No declaration of access: default access
    - allow access by other classes in the same package

Inheritance

- Inheritance:
  - Allows a class to share members defined from another class.
  - Define a new class from an existing class.
- Inherited class is called Sub-class or derived class
- The class for inheritance is called the base class or super class
- Multiple inheritance - is implemented on interfaces.

Inheritance.. Subclass

- Use **extends** keyword to define subclasses
  - public class MyPoint extends Point
- Sub-class
  - Inherit all members of the base class
  - Direct access to private members of the base class is not allowed
  - may have members with the same name as the base class
    - use keywords **super** to refer to the direct superclass of a class
    - Use **this** to refer to the class itself
  - inherited from only 1 base class - single inheritance

Inheritance.. subclass constructor

- The subclass constructor must initialize data members that inherited from the superclass, by calling the superclass constructor –
  - use the syntax **super(argument);**
  - call to super must be the first statement in the subclass's constructor.
  - the compiler will automatically call the default constructor of the parent class if the subclass does not invoke–> all classes should have default constructor
-
```
class Point2D {
 private int x,y;
 public Point2D() { x=y=0;}
public Point2D(int a, int b) { x=a; y=b;}
//..other components
}
```
-
```
class Point3D extends Point2D {
 private int z;
 public Point3D(int x, int y, int z) {
super(x,y); this.z=z;
 }
}
```

Polymorphism

- Polymorphism
  - is the feature that a task can produces different effects depend on referencing context.
  - allow multiple methods with the same name in the same class or in classes with inheritance relations
- Two mechanisms of polymorphism in Java:
  - overloading method
  - overriding methods
- Helps improve code organization, extensibility adds new features without affecting old components.

Polymorphism.. overloading

- Overloading:
  - Define methods with the same name in the same class
  - must have argument lists that differ in type or order
  - regardless of the difference in return type, access scope of overloaded methods
- Example:
  - public int max(int x, int y, int z)
  - public void max(int a,int b, int c)
  - private double max(double a, double b, double c)
  - private double max(double a, double b)
  - public int max(int a, int b)

Polymorphism.. overriding

- overriding
  - Re-define methods of super-class in sub-class
  - Override method has same signature with overridden method
  - Access scope should not be narrow as compared to base class method
  - method in subclass xxx() can invoke overridden method xxx() in parent class by **super.xxx().**
  - Late binding mechanism – link source code at runtime
- Eg
  - The Object class is the default base class of all other classes
  - **public String toString()** of the Object class is overriden by all other classes

Compare overloading & overriding

- Overloaded methods:
  - additions
  - unlimited quantity
  - arguments must be different
  - return type may be different
- Overriding methods:
  - replacement.
  - only one
  - arguments must be the same
  - return type must be the same

Dynamic binding

- Binding is the process of linking the executable code with the function call.
  - Static linking: binding is determined at compile time
  - dynamic binding: binding is determined at run time
- In Java, binding takes place at run time - depending on the reference object that invokes the method - not the type of the variable. This mechanism is called -late binding or -dynamic binding
  - Dynamic binding is the basis of polymorphism in OOP
  - The instanceof operator allows to check the actual class of the object at run time.

OOP Design

- Modularization: is the process of decomposing a problem into a set of modules to reduce the overall complexity of the problem.
- Hierarchy - hierarchical: create related sub-systems until the most basic components are reached.
  - IS – A: inheritance type relationship
    + a Rose is a Flower
    + class Rose extends Flower
  - HAS – A : aggregation relationship– PART OF
    + a Car has 4 wheels, brake..
    + member of a class are other classes..
- Reusability of design through creation of new classes by adding features to existing classes
- Generalization – Specialization
- Aggregation or Composition

Practice Lab: Chapter 3

- Write IntArray class in OOP . style
  - Draw class diagram
  - Every operation on the array is replaced by a method
  - Write main() from another class to run program
- Assignment 1: Virtual Shop
  - Individual asignment
  - See virtualshop.doc description

Constructive Questions.

- Compare the advantages and disadvantages of object-oriented programming and functional programming.
- Indicate the objects of the human resource management program according to your assumptions and understanding.
- Find examples of classes with static properties.
- The counter variable i is used in most of the class functions, where should the variable i be declared?
- How many constructors should a class have? Declare at least 5 constructors of class Student(id, name, dob, tel, password)
- Declare a method that allows to recover forgotten password of Student class? Assume the conditions for the pass to be reissued.
- Point out the similarities of the base class-subclass relationship in OOP programming and the relationship in a family

Constructive Questions.

- Design the classes of a hospital's cost management system according to your understanding and reasonable assumptions.
- Find practical examples that illustrate the need to use protected keyword when declaring class members.
- What if a class declares all its members private?
- Compare override and overload characteristics of polymorphism
- Design the Medicine class - representing the drugs in the management system of a pharmaceutical company - according to your knowledge.
- Designing classes of online sales problems, applying modularity and hierachy ideas.