Lecture4 – OOP concepts in Java

- Object Oriented Programming in Java
  - Review
    - Concept of class/object
      - Constructors
  - Inheritance
  - Data encapsulation
  - Polymorphism

Inheritance

Inheritance

- Let's take the account example
- There can be many types of accounts
  - Checking, saving, money market, IRA, etc.
- All accounts may have
  - Some common members.
    - Account number, user SSN, etc.
  - Some class specific members.
    - Checks cleared, investment options, etc.
- Method implementation may be
  - Same in different classes
  - Different in different classes.

Inheritance – base class & derived class

- Base class or parent class

```
class account
{
    private int user_SSN;
    private int accountNumber;
    public account ( ) { .. }
    public void deposit (int amount)    { …  }
    public void withdraw (int amount) { …  }
};
```

- Derived class or child class

```
class checkingAccount  extends account     // checkingAccount is
{                                          // derived from account
    private int lastCheckCleared;          // not present in account
    public  void showAllChecksCleared( ) { } // not present in account
};
```

Inheritance – base class and derived classes

- Base Class

```
class account
{
   private int user_SSN;
   private int accountNumber;
   public  account ( ) { … } // code
   public  void deposit (int amt)
    {
      // code
    }
   public  void withdraw (int amt)
    {
      // code
    }
};
```

- Derived (or child) class-1

```
class checkingAccount extends account
{
     private int lastCheckCleared;
     public checkingAccount ( ) { … };
     public void showChecksCleared ( ) { //code
      }
};
```

- Derived (or child) class-2

```
class IRA_account extends account
{
     public IRA_Account ( ) { … };
     public void buyFund (int fund_ID)  {
     //code
      }
     public void sellFund (int fund_ID)  {
     //code
      }
};
```

Inheritance – continued.

- Important points to note:
  - Derived classes have access to public members of base classes in this example.
  - Derived classes can have their own members.
    - E.g. showLastCheckCleared( ), buyFund( ), sellFund( ), etc.
    - Members of one derived class are not accessible to another.

Examples

- Valid usage in an external function
  - Account acct = new Account ( );
  - checkingAccount ca = new checkingAccount ( );
  - acct.deposit (700);
  - acct.withdraw (300);
  - checkingAccount.deposit (1000);
  - checkingAccount.withdraw (600);
- Invalid usage in an external function
  - acct.user_SSN = 1234; // Can't access user_SSN
  - acct.accountNumber = 567;

Inheritance … Object class

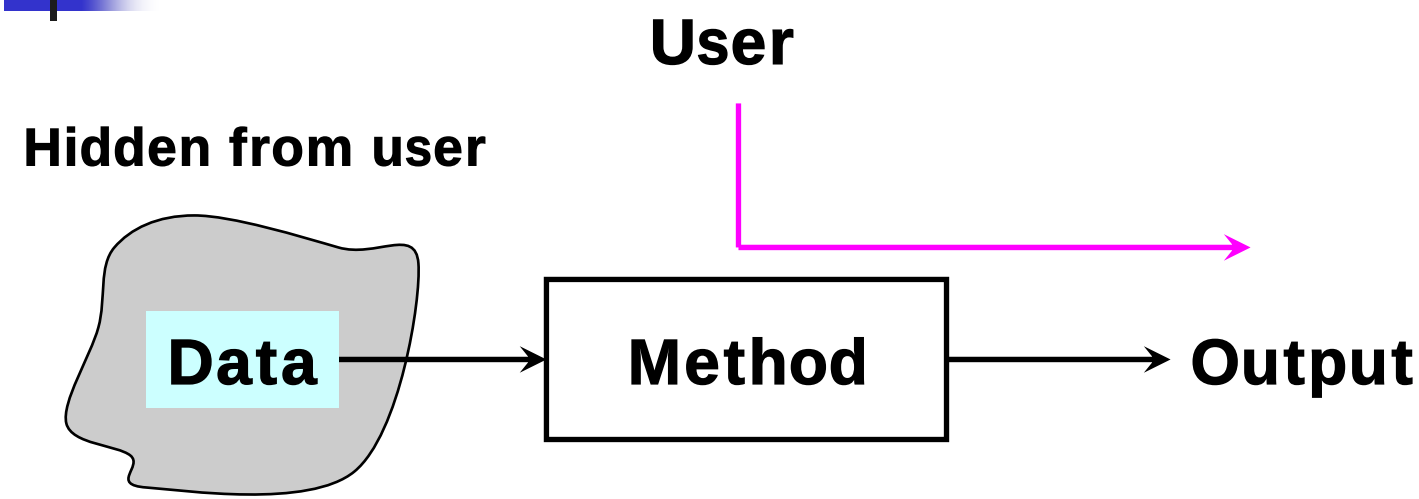
- In Java Object is the base class for every Java class.
- Object is a built-in class.
- Defines useful functions.
  - hashCode
  - toString
  - equals
  - notify, etc.

Data encapsulation – review

Data encapsulation

- Hide the data from other classes
- Need to know what methods are implemented
- Not how they are implemented
- Provide interfaces (APIs) to access data
- E.g. To compute interest in a bank an user
  - Needs to know what function to call
  - NOT how the function is implemented

Data encapsulation … contd.



- Methods act on data to provide output.
- User needs to see only method, not data.
- User should not be affected by
  - Implementation details of methods.
  - Changes in implementation of methods.

Data encapsulation … contd.

- Not all data needs to be hidden
  - It is fine to give direct access to some data.
- Not all methods need to be given access
  - Some methods may be hidden – for internal use by classes

⇒ Data and methods both need access restrictions.

- How can data/methods be hidden?
  - By using access modifiers.
- Different access modifiers:
  - public – accessible to every class, function
  - private – accessible only to class
  - protected – accessible to class package and subclass
  - No modifier – accessible only to class and package

Access modifiers

Modifier	class	package	subclass	Other classess
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
No modifier	Yes	Yes	No	No
private	Yes	No	No	No

Source: Oracle.com

Data encapsulation in account example

In an object of account

- user_ssn and accountNumber are declared private
  - Accessible only to account and nothing else.
  - Methods are public
    - Anyone can access them.

```
public class Account
{
  private  int user_SSN;                          // Accessible only to Account
  private  int accountNumber;                     // Accessible only to Account
  public   Account ( ) { .. }                          // Accessible to all
  public   void withdrawMoney (int amount) { .. } ; // Accessible to all
  public   void depositMoney (int amount)   { .. }; // Accessible to all
  public   void computeInterest( )          { .. }; // Accessible to all
 …
};
```

Polymorphism

Polymorphism

- Poly – many, Morphism – ability to take many forms
  - Ability of objects to behave differently
  - Achieved by using different implementations of the same function in different classes.
  - Parent class defines and implements a function in one way.
  - Child classes can override the function.

Polymorphism

```
public class Account
{
    public Account() { }

    public void showAccountType  ( )
    {
        System.out.println ("Account");
    }

    public static void main (String args[ ])
    {
       Account a  = new Account();
       Account ca = new CheckingAccount( );
       Account sa = new SavingsAccount( );
       a.showAccountType( );  //Account
       ca.showAccountType( ); //CheckingAccount
       sa.showAccountType( ); //SavingsAccount
    }
};
```

```
class CheckingAccount extends Account
{
    public CheckingAccount() { }

    public void showAccountType  ( )
    {
       System.out.println ("CheckingAccount");
    }
};

class SavingsAccount extends Account
{
    public SavingsAccount( ) { }

    public void showAccountType  ( )
    {
        System.out.println ("SavingsAccount");
    }
};
```

Polymorphism … contd.

- In the previous example
  - a, ca and sa are defined of type Account
  - But they each executed a different showAccountType function.
    - a executed the function in class Account
    - ca executed the function in class CheckingAccount
    - sa executed the function in class SavingsAccount.
  - Reason this is possible
    - Each object is created differently
    - a is created as Account, ca as CheckingAcocunt, sa as SavingsAccount
    - This is an example of late binding or runtime binding
    - At runtime, objects are bound to the correct type and the corresponding function is executed.

Executing a member function

- In any class, when a member function is called,
  - The member function of the most specific class is executed.
- E.g. object o is created of type class c
- If a member function o.f( ) is called, function in c is executed, if it exists.
- Otherwise, the function f( ) in the closest parent in the hierarchy is executed.

Overloaded functions

- A function with the same function name
  - With different arguments
    - Same number of arguments, but different types
    - Different number of arguments
- E.g.

```
class foo
{
   void overloadedFn(int a) {… };
   void overloadedFn(String s) {…};
   void overloadedFn( ) {…};
   void overloadedFn(int a, double b) {…};
}
```