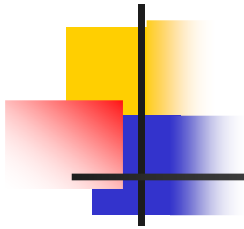
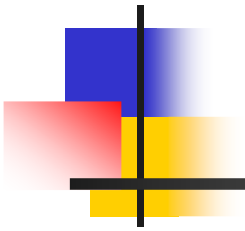
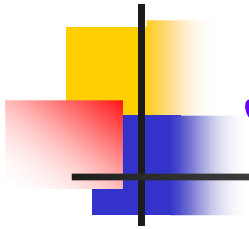


Lecture3

- Functions
- OOP concepts of Java
  - class and object
  - Data abstraction
    - public and private members
  - Inheritance



Functions



Java functions

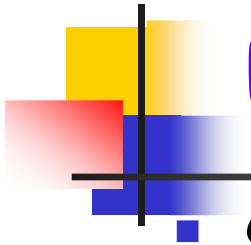
- A group of statements
  - To perform a task
  - Possibly return a value
- Unlike C, functions are part of a class in Java

- Syntax

```
<return_type> fn_name (arguments)
    {
        // function body
    }
```

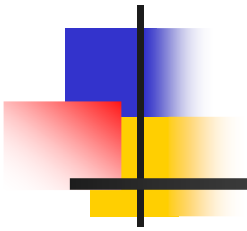
Java functions … example

```
import java.io.*;

public class anyClass                          // Define a class first
{
   int square (int x)                          // function to compute square
     {
        return (x * x);
     }

   public static void main(String args[ ])  //Starting point of the program
     {
        int i = 5;
        anyClass ac = new anyClass();          // Create an object first
        int i_sq = ac.square (5);              // Call its function
        System.out.println ("Square of 5 is: " + i_sq);
     }
}
```

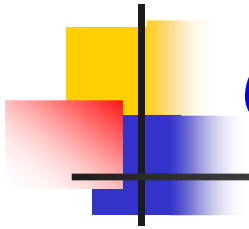


Useful Java functions

- clone ( )
  - Create a copy of an existing object
- equals ( )
  - Checks if two objects are the same
  - This is not quite the same as == operator
- finalize ( )
  - Called to clean up object's resources.
- getClass ( )
  - Returns a class object
- hashCode ( )
  - Returns object's memory address in hexadecimal
- toString ( )
  - Returns a string representation of the object.

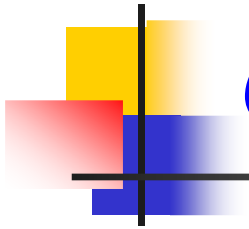


OOP - Class and objects



Class and objects

- class – the basic unit of OOP in Java
- A class typically corresponds to some meaningful entity.
- class has both data and methods.
- Attributes and methods are members of a class
- An instance of a class is an object.
- A class uses methods to interact with other classes/functions.



Class and objects … contd.

- Classes may have both
  - Data attributes
    - Can be of basic or user defined data types.
    - Need to be initialized – typically done in a constructor
  - Methods
    - Functions that are part of classes
    - Typically interfaces to interact with other classes and functions.
    - Provide APIs to external world to access and manipulate data attributes.

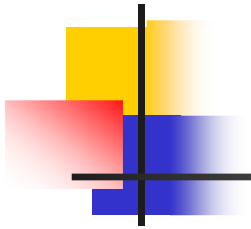
Constructor and destructor … contd.

Constructor

- A function with the same name as the class
- Called when an object is created
- A class can have more than one constructor

Destructor

- There is NO destructor in Java, equivalent to C++ destructor
- In C++
  - Destructor: A function with the name ~classname( )
  - Called when the object goes out of scope, or deleted.
- In Java, closest equivalent is finalize( ) function
  - Used to clean up system resources
    - E.g. close open files, open sockets
    - Clear screen for GUI/graphics objects.
  - Called by system garbage collectors and other resource cleanup functions.



A simple "account" example

```
public class Account
{
  private  int user_SSN;                                  // attribute (data)
  private  int accountNumber;                             // attribute (data)
  public   void withdrawMoney (int amount) { .. } ;       // method
  public   void depositMoney (int amount)   { .. };       // method
  public   void computeInterest( )          { .. };       // method

  public Account() { }                                    // Constructor
  public static void main (String args[ ])                // main function
    {
       Account a = new Account( );                        // Create a new object
       System.out.println ("In account main");
    }
};
```

Account example … contd.

```
import java.io.*;

public class Account
{
  private  int user_SSN;              // attribute (data)
  private  int accountNumber;         // attribute (data)
  public   myClass m = null;

  // Account constructor
  public Account( )
    {
       user_SSN = -1; accountNumber = -1; // default values
    }

  // Account finalize function
  protected void finalize( )
    {
       System.out.println ("In Account finalize function");
    }

  // Account main function
  public static void main (String args[ ])
    {
       Account a = new Account( );
       System.runFinalizersOnExit(true); // deprecated
    }
};
```

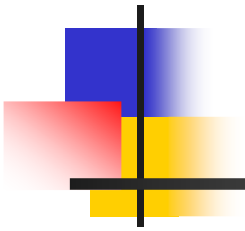
Class definition

Constructor

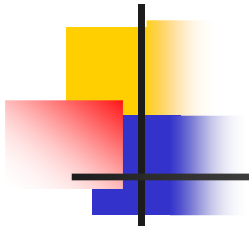
finalize function

main function

Calls finalize functions



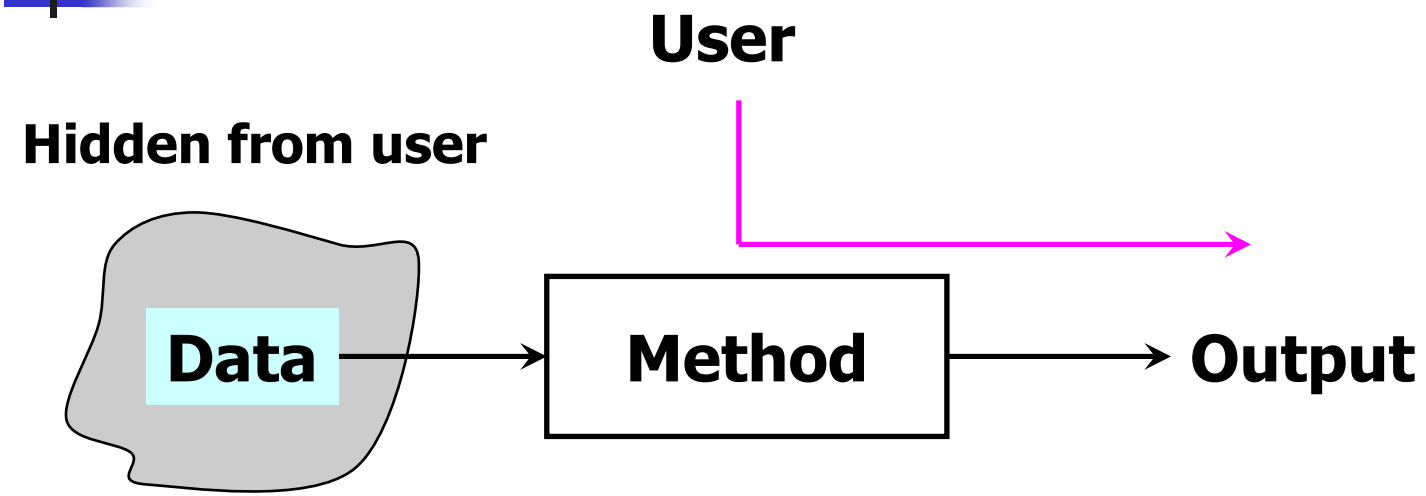
OOP – Data encapsulation



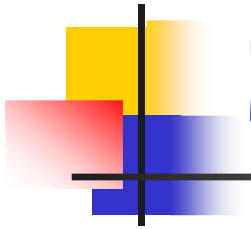
Data encapsulation

- Hide the data from end user
- 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.

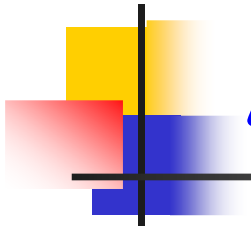


Data encapsulation … contd.

- 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.

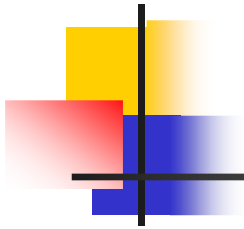
- How can data/methods be hidden?
  - By using access modifiers.
- Different access modifiers:
  - public – accessible to every class, function
  - private – accessible only to class and package
  - protected – accessible to class package and subclass
  - No modifier – accessible only to class and package



Access modifiers

Modifier	class	package	subclass	others
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
No modifier	Yes	Yes	No	No
private	Yes	No	No	No

Source: Oracle.com

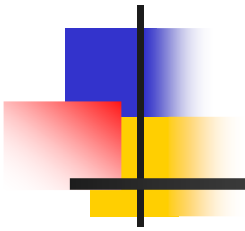


Data encapsulation in account example

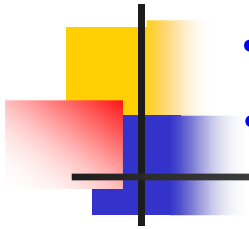
In an object of account

- 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
 ...
};
```



Inheritance



Inheritance

- Let's take the account example again
- 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

```
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

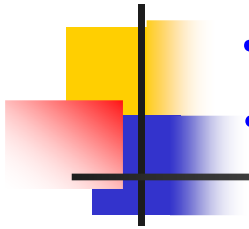
```
class account
{
   private int user_SSN;
   private int accountNumber;
   public  Account ( ) { … } // code
   public  void deposit (int amt)
    {
      // code
    }
   public  void withdraw (int amt)
    {
      // code
    }
};
```

- 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 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.