Chapter 1

Object-Oriented Programming Using Java

Objectives

Discuss the following topics:

- Rudimentary Java
- Object-Oriented Programming (OOP) in Java
- Input and Output
- Java and Pointers
- Vectors in `java.util`
- Data Structures and Object-Oriented Programming
- Case Study: Random Access File

Rudimentary Java

compile language must compile first

- A **Java program** is a sequence of statements that have to be formed in accordance with the predefined syntax
- A **statement** is the smallest executable unit in Java เรียกใช้งาน sentence
- Each statement ends with a semicolon (";")
- **Compound statements**, or **blocks**, are marked by delimiting them with braces, { and } ชุดคำสั่งภายใน { }

Variable Declarations

- Each variable must be declared before it can be used in a program
- It is declared by specifying its type and its name
- Variable names are strings of any length of letters, digits, underscores, and dollar signs that begin with a letter, underscore, or dollar sign
- A letter is any Unicode letter
- Java is case sensitive

Variable Declarations (continued)

- A type of variable is either:
  - One of the eight built-in basic types
  - A built-in or user-defined class type
  - An array

Variable Declarations (continued)

Table 1-1 Variable built-in types and their sizes

Type	Size	Range
`boolean`	1 bit	`true, false`
`char`	16 bits	Unicode characters
`byte`	8 bits	[-128, 127]
`short`	16 bits	[-32768, 32767]
`int`	32 bits	[-2147483648, 2147483647]
`long`	64 bits	[-9223372036854775808, 9223372036854775807]
`float`	32 bits	[-3.4E38, 3.4E38]
`double`	64 bits	[-1.7E308, 1.7E308]

Operators

- Value assignments are executed with the assignment operator =
- Use one at a time or string together with other assignment operators

```
x = y = z = 1;
```

- For a **prefix operator**, a variable is incremented (or decremented) first and then an operation is performed in which the increment takes place
- For a **postfix operator**, autoincrement (or autodecrement) is the last operation performed

Decision Statements

- One decision statement is an `if-else` statement

```
if (condition)
    do something;
[else do something else;]
```

- A `switch` statement is shorthand for nested `if` statements

```
switch (integer expression) {
    case value1: block1; break;
    . . . . . .
    case valueN: blockN; break;
    default: default block;
}
```

Loops

- The first loop available in Java is the `while` loop:

  `while` (*condition*)
  do something;

- The second loop is a `do-while` loop:

  `do`
  do something;
  `while` (*condition*)*;*

- The third loop is the `for` loop:

  `for` (*initialization*; *condition*; *increment*) (int i = 0; i ≥ 10; i++)
  do something;

Exception Handling

- Catching an error is possible by using the `try-catch` statement

```
try {
    do something;
} catch (exception-type exception-name) {
    do something;
}
```

- The number of catch clauses is not limited to one

throw and catch

```
public int f1( int [ ]  a, int  n) throws ArrayIndexOutOfBoundsException {
     return  a[n]  +  a[n+1] ;
}
```

```
public void f2( ) {
  int [ ] a = {1,2,3,4,5};
   try {  for (int i = 0; i<a.length;i++)
                System.out.println(f1(a,i) + “  "  );
   } catch (ArrayIndexOutOfBoundsException e) {
          System.out.println(“Exception caught in f2() “);
          throw e;
    }
}
```

```
public void f3( ) {
  try {   f2( );
  } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(“Exception caught in f3( )” );
  }
}
```

Object-Oriented Programming (OOP) in Java

- A **class** is a template in accordance to which objects are created
- Functions defined in a class are called **methods**
- Variables used in a class are called **class scope variables**, **data fields**, or **fields**
- The combination of data and related operations is called **data encapsulation** * hiding data — เรียกอีก class จะใช้ได้โดยไม่รู้ info ภายใน
- An **object** is an instance of a class, an entity created using a class definition

Encapsulation

- Objects make the connection between data and methods much tighter and more meaningful
- The first OOL was Simula; it was developed in the 1960s in Norway
- The **information-hiding principle** refers to objects that conceal certain details of their operations from other objects so that these operations may not be adversely affected by other objects

Class Methods and Class Variables

- Static methods and variables are associated with the class itself and are called **class methods** and **class variables**
- Nonstatic variables and methods are called **instance variables** and **instance methods**
- The method `main()` must be declared as `static`

Generic Classes

```
class IntClass {
       int[] storage = new int[50];
       ...................
}

class DoubleClass {
       double[] storage = new double[50];
       ...................
}

class GenClass {
       Object[] storage = new Object[50];
       Object find(int n) {
           return storage[n];
       }
       ...................
}
```

could be anything ex. int, double

defind sth that don't know type =>"Object"

Generic Classes

```
class GenClass2<T1, T2> {
        T1 t11, t12;
        T2 t2;
        GenClass2(){
            ..........
        }
        T1 method1(T1 t){
            t11 = t;
            ..........
            return t12;
        }
        ........
}
```

constructor

```
   GenClass2<Integer,Double> ob1=new
   GenClass2<Integer,Double>();
GenClass2<String,String> ob2 =new GenClass2<String,String>
   ();
```

Assignment 1 Due Jan/~~28~~ 27/22

- Programming Assignments in Chapter 1 Problem 1 (page 56)
  - Fraction Class
  - Constructor(s)
  - Methods
    - toString() to print the fraction such as 1/2, 7/13
    - add(), subtract(), multiply(), divide()
      - 5/6 * 3/8 = 15/48
      - 5/6 + 3/8 =
    - reduce() For example 12/18 = 2/3.
- Java document: http://java.sun.com/j2se/1.5.0/docs/api/allclasses-noframe.html

Object Oriented Programming

Discuss the following topics:

- Object-Oriented Programming (OOP) in Java
- Input and Output
- File Processing

Object-oriented programming

A *class* in Java is a software construct that includes

- *fields* (also called *data fields* or *class scope variables*) to provide data specification, and
- *methods* which are functions operating on these data and possibly on the data belonging to other class instances.

This combining of the data and related operations is called *data encapsulation.*

An *object* is an instance of a class, an entity created using a class definition.

Object-oriented programming

Classes correspond to concepts defined in terms of common properties and common behavior, objects corresponds to actualization of these concepts.

Country class:

- name
- population
- area
- loan
- sender
- compute population density
- get international loan

Country objects:

Classes and objects in Java

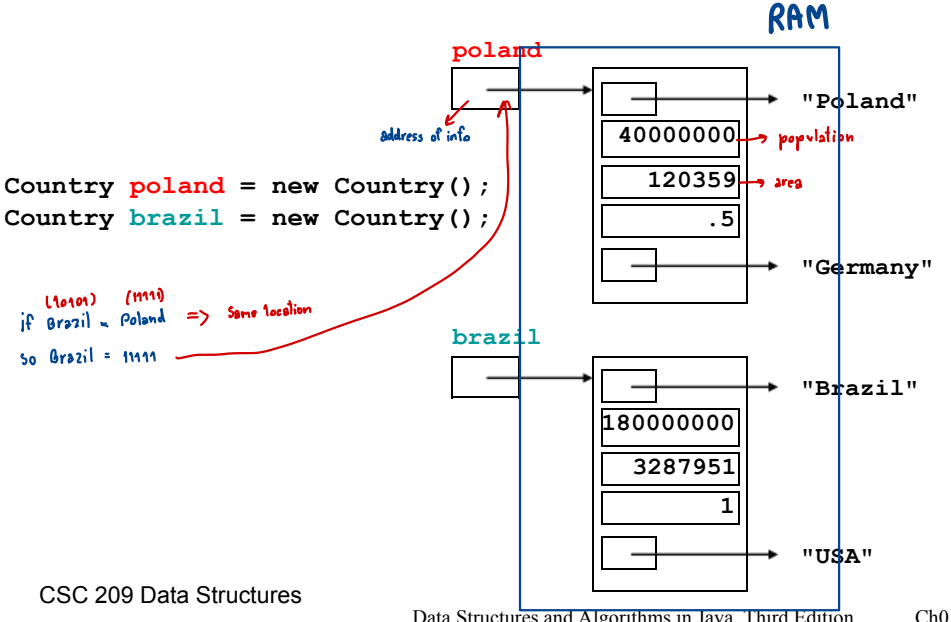
```
class Country {
   String name;
   long population;
   double area;
   double loan;
   String sender;
   double density() {
       return area/population;
   }
   void internationalLoan(double ln, String from) {
       loan = ln;
       sender = from;
   }
   . . . . . .
}

class Countries {
   Country poland = new Country();
   Country brazil = new Country();
}
```

data fields

methods

objects

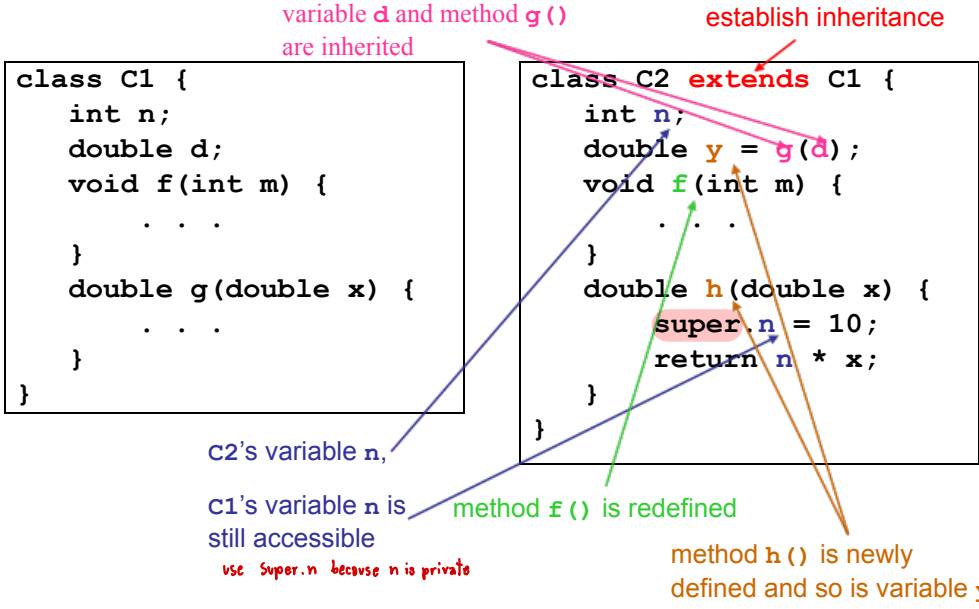
Objects in Java



Inheritance (extend)

A new class can be derived form an existing class whereby the new class automatically includes (inherits) fields and methods of the existing class. The former is called a *subclass* or *derived class*, the latter is called a *superclass* of *base class*.

Inheritance



Accessibility control

- The **`private`** modifier indicates methods and fields that can be used only by this class
- The **`protected`** modifier means that a method or a data field is accessible to derived classes and in the package that includes the class that declares the method or the data field.
- A default modifier is no modifier at all, which indicates access to methods and fields in the package that includes the class that declares the methods or the data fields.
- Methods and fields declared **`public`** can be used by any other object.

Accessibility control

place of access	`private`	`protected`	no modifier (≠ C++)	`public`
same class	yes	yes	yes	yes
same package subclass	no	yes	yes	yes
same package non-subclass	no	yes (≠ C++)	yes	yes
different package subclass	no	yes	no	yes
different package non-subclass	no	no	no	yes

```
class C1 {
 private int k = 11;
 protected int m = 12;
 int n = 13; // package
 public int p = 14;
}
```

```
class C2 extends C1 {
 C2() {
//  k = 21;      ← k is private
    m = 22;
    n = 23;
    p = 24;
 }
}
```

```
class C4 {
 void f() {
    C1 c1 = new C1();
//  c1.k = 41;  ← k is private
    c1.m = 42;
    c1.n = 43;
    c1.p = 44;
 }
}
```

← two packages

↓

```
class C3 extends C1 {
 C3() {
//  k = 31;      ← k is private
    m = 32;
//  n = 33;      ← n is in different package
    p = 34;
 }
}
```

```
class C5 {
 void f() {
    C1 c1 = new C1();
//  c1.k = 51;  ← k is private
//  c1.m = 52;  ← m is protected
//  c1.n = 53;  ← n is in different package
    c1.p = 54;
 }
}
```

Arrays

- Arrays are Java objects
- There is no keyword (as an object name) with which all other arrays are declared
- Without keywords, subclasses cannot be created
- Arrays are declared with empty brackets after the name of the type or the name of the array itself
- These two declarations are equivalent:

```
int[] a;
```
and
```
int a[];
```

Wrapper Classes

- A declaration of a basic data type also creates an item of the specified type
- **Casting** converts a value to a different type.

```
double price = 25.75;
int baht = (int) price;
int stang = (int)  ((price – baht)*100);
```

- **Wrapper classes** provide object versions of basic data types

```
int p=9; double q = 20.5;
System.out.println(p + Integer.toOctalString(p) );
```

Abstract Data Types (ADT)

- An item specified in terms of operations is called an **abstract data type**
- In Java, an abstract data type can be part of a program in the form of an interface
- **Interfaces** are similar to classes, but can contain only:
  - Constants (**final** variables)
  - Specifications of method names, types of parameters, and types of return values

Abstract Data Types (continued)

```
interface I {
       void Imethod1(int n);
       final int m = 10;
}
class A implements I {
       public void Imethod1(int n) {
           System.out.println("AIf1 " + n*m);
       }
}
abstract class AC {
       abstract void ACmethod1(int n);
       void ACmethod2(int n) {
           System.out.println("ACf2 " + n);
       }
}
class B extends AC {
       public void ACmethod1(int n) {
           System.out.println("BACf1 " + n);
       }
}
```

Inheritance

- OOLs allow for creating a hierarchy of classes so that objects do not have to be instantiations of a single class
- **Subclasses** or **derived classes** inherit the fields and methods from their base class so that they do not have to repeat the same definitions
- A derived class can override the definition of a `non-final` method by introducing its own definition

Polymorphism

- **Polymorphism** is the ability of acquiring many forms
- **Dynamic binding** is when the type of method to be executed can be delayed until run time
- **Static binding** is when the type of response is determined at compilation time
- **Dynamic binding** is when the system checks dynamically the type of object to which a variable is currently referring and chooses the method appropriate for this type

Polymorphism (continued)

```
class A {
       public void process() {
           System.out.println("Inside A");
       }
}
class ExtA extends A {
       public void process() {
           System.out.println("Inside ExtA");
       }
}
If the Code: A ob = new A();  ob.process();
             ob = new ExtA(); ob.process();
Result???
```

Static and dynamic binding

- Java checks the type of object to which a reference is made and chooses the method appropriate for this type at compilation time producing static binding during run time producing dynamic binding.

- *Polymorphism* is an ability to associate with the same method name different meanings through the mechanism of dynamic binding.

Dynamic binding

```
class A {
   public void process() {
        System.out.println("Inside A");
   }
   void f(A a) {
        a.process();
   }
}
class ExtA1 extends A {
   public void process() {
        System.out.println("Inside ExtA1");
   }
}
class ExtA2 extends A {
   public void process() {
        System.out.println("Inside ExtA2");
   }
}
```

```
A ob = new A();
ob.process();
ob = new ExtA1();
ob.process();
ob = new ExtA2();
ob.process();
```


```
Inside A
Inside ExtA1
Inside ExtA2
```

Dynamic binding

```
class A {
   public void process() {
        System.out.println("Inside A");
   }
   void f(A a) {
        a.process();
   }
}
```

```
A ob = new A();
A a1 = new A();
ExtA1 a2 = new ExtA1();
ExtA2 a3 = new ExtA2();
ob.f(a1);
ob.f(a2);
ob.f(a3);
```


```
Inside A
Inside ExtA1
Inside ExtA2
```

Polymorphism (continued)

```
Student s = new Student(“Tom”,2.56);
Employee e = new Employee(“Mark”,”Teacher”);
Person p = null;
If (….)
  p = s;        //Suppose Student extended from Person
else
  p = e;       //Suppose Emploee extended from Person
System.out.println(“Person is” + p.toString());
```

Exercise 1 (Paper) Due Wed/Feb/2/22

Create a class for Banking Account composed of account number, balance, 100 Transactions

Create Saving and Checking Classes:

Saving account maintains an interest rate.

Checking account maintains an overdraft amount.

Create a class for Transaction

Transaction number, date, amount (positive for deposit, negative for withdraw)

Write your answer in a piece of paper.

Input and Output

- To print anything on the screen, use the statements:

```
System.out.print(message);
System.out.println(message);
```

- To read from Scanner class

```
import java.util.Scanner;
Scanner in = new
Scanner(System.in);
```

- The methods of Scanner include next(), nextbyte(), nextint(), nextline(), nextlong(), etc.

```
in.nextInt();
```

Reading from / Writing to File

- To use the classes for reading and writing data, the `java.io` package has to include the statement:

```
import java.io.*;
```

- To read one line at a time, use the method `readLine()` from `BufferedReader` which requires reader such as `FileReader()` for a parameter.

Reading from / Writing to File

```
FileReader f = new FileReader("MyFile");
BufferedReader fin = new  BuferredReader(f);
fin.readLine();
```

or

```
BufferedReader fin = new BufferedReader(new FileReader("MyFile"));
```

Similar to writing

```
BufferedWriter fout = new BufferedWriter(new FileWriter("MyFile"));
fout.write("Hello");
```

Do not forget

```
fin.close();        and   fout.close();
```

Reading Tokens: Words and Numbers

- The `nextToken()` method skips space characters separating tokens and updates the tokenizer's instance variables:
  - `sval` of type `String`, which contains the current token when it is a word
  - `nval` of type `double`, which contains the current token when it is a number
  - `ttype` of type `int`, which contains the type of the current token

Reading Tokens: Words and Numbers (continued)

- There are four types of tokens:
  - `TT_EOF` (end of file)
  - `TT_EOL` (end of line)
  - `TT_WORD`
  - `TT_NUMBER`

Reading Tokens: Words and Numbers (continued)

```
void readTokens(String fInName) throws IOException {
       StreamTokenizer fIn = new StreamTokenizer(
                         new BufferedReader(
                         new FileReader(fInName)));
       fIn.nextToken();
       String s;
       while (fIn.ttype != StreamTokenizer.TT_EOF) {
           if (fIn.ttype == StreamTokenizer.TT_WORD)
               s = "word";
           else if (fIn.ttype ==
  StreamTokenizer.TT_NUMBER)
               s = "number";
           else s = "other";
           System.out.println(s + ":\t" + fIn);
           fIn.nextToken();
       }
}
```

Reading and Writing Primitive Data Types

- The `DataInputStream` class provides methods for reading primitive data types in binary format
- The methods include:
  - `readBoolean()`
  - `readByte()`
  - `readShort()`
  - `readChar()`
  - `readInt()`
  - `readLong()`
  - `readUTF()` (to read strings in Unicode Text Format)

File Processing and Java Utils

Discuss the following topics:

- I/O
- File Processing
- Random Access File
- Java and Pointers
- Vectors in `java.util`

Random Access Files

- To be able to both read and write in the same file at any position in the file, a **random access file** should be used
- A file is created with the constructor:

```
RandomAccessFile(name, mode);
```

- The constructor opens a file with the specified name either for reading, or for reading and writing:

```
RandomAccessFile =
    raf new RandomAccessFile("myFile", "rw");
```

Random Access Files (continued)

- The method `length()` returns the size of the file measured in bytes
- The method `getFilePointer()` returns the current position in the file
- The method `seek(pos)` moves the file pointer to the position specified by an integer `pos`

Random Access Files (continued)

- Reading is done by:
  - `read()`, which returns a byte as an integer
  - `read(b)`, which fills entirely a byte array `b`
  - `read(b,off,len)`, which fills `len` cells of the byte array `b` starting from cell `off`
  - `readLine()`, which reads one line of input

Java and Pointers

- Although Java does not use explicit pointers, object access is implemented in terms of pointers
- In many languages, **pointer** is a technical term for a type of variable; in Java, the term **reference** is used instead

Java and Pointers (continued)

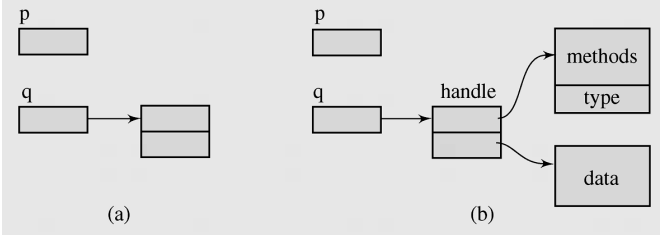


Figure 1-1 Object reference variables p and q
(a) logic of reference of q to an object
(b) implementation of this reference

Java and Pointers (continued)

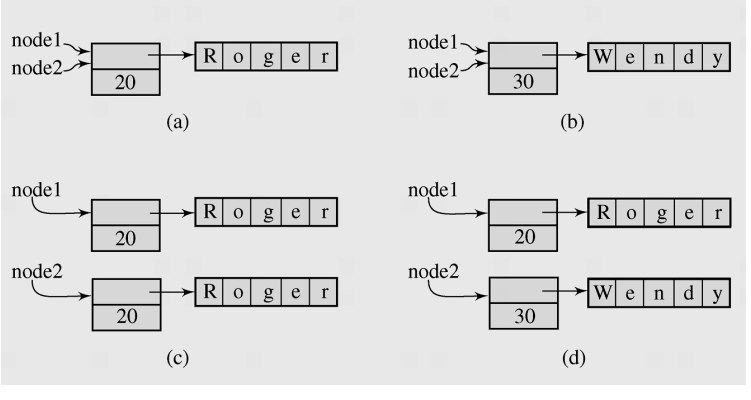


Figure 1-2 Illustrating the necessity of using the method `clone()`

Vectors in `java.util`

- A **vector** is a data structure with a contiguous block of memory, just like an array
- Class `Vector` is a flexible array whose size can be dynamically changed
- The class hierarchy in the package `java.util` is:

```
Object → AbstractCollection → AbstractList →
Vector
```

Vectors in `java.util` (continued)

- The status of the vector can be tested with two methods:
  - `size()`, which returns the number of elements currently in the vector
  - `capacity()`, which returns the number of cells in the vector
- If the vector's capacity is greater than its size, then a new element can be inserted at the end of the vector immediately

Vectors in `java.util` (continued)

- The method `trimToSize()` should be used to reduce wasted space
- The method `ensureCapacity()` should be used to set the maximum number of elements inserted in a vector
- The method `ensureCapacity()` affects only the capacity of the vector, not its content
- The method `setSize()` affects its content and possibly the capacity

Vectors in `java.util` (continued)

- The method `addElement()` adds an element at the end of the vector
- The insertion of an element in any other position can be performed with the method `insertElementAt()`
- The method `elements()` puts vector elements in an object of `Enumeration` type
- The method `clone()` clones the array implementing the vector, but not the objects in the array

Programming 2
Due Feb/9/2021

- Programming Assignments in Chapter 1 Problem 5 (page 57)

Index -Vector	
Data Value	Position
Jack	10
Jim	30
John	20
Sack	50
Tom	0
Tony	40

FILE
Tom
Jack
John
Jim
Tony
Sack

Vector

City	Location
Bangkok	770
Bangmod	0
Thungkru	385

city.dat in Random Access File

City	Lat	Long	Country	ISO2	ISO3	Admin_city	type	population
150	double	Double	50	2	3	150	Double	Int
150	+8	+8	+50	+2	+3	+150	+10	4 = 385
Bangmod	104.22	10.25	Thailand	TH	THA	Thungkru		7562145
Thungkru	385+150					Bangkok	minor	
Bangkok								385*2 + 381

Data Structures and Object-Oriented Programming

- The data structures field is designed for:
  - Building tools to be incorporated in and used by programs
  - Finding data structures that can perform certain operations speedily without imposing too much burden on computer memory
  - Building classes by concentrating on the mechanics of these classes
  - Investigating the operability of these classes by modifying the data structures to be found inside the classes

Summary

- A Java program is a sequence of statements that have to be formed in accordance with the predefined syntax.
- A statement is the smallest executable unit in Java.
- Compound statements, or blocks, are marked by delimiting them with braces, { and }.
- A class is a template in accordance to which objects are created.

Summary (continued)

- Functions defined in a class are called methods.
- Variables used in a class are called class scope variables, data fields, or fields.
- The combination of data and related operations is called data encapsulation.
- An object is an instance of a class, an entity created using a class definition.
- An item specified in terms of operations is called an abstract data type.

Summary (continued)

- Subclasses, or derived classes, inherit the fields and methods from their base class so that they do not have to repeat the same definitions.
- Polymorphism is the ability of acquiring many forms.
- In many languages, pointer is a technical term for a type of variable; in Java, the term reference is used instead.
- A vector is a data structure with a contiguous block of memory, just like an array.

Case Study: Random Access File

- Write a generic program that generates a random access file for any type of record
- Each record consists of five personal fields:
  - ID, name, city, year of birth, and salary
- And a student file that consists of the personal fields and academic major

Design

- Class
  - Person to maintain a record of a person
    - Subclass Student extended from Person
  - Database to maintain files for records
    - DB for persons
    - DB for students
  - Main to switch between two databases
- Interface “DBobject” for class Person
  - Person must be implemented from the DBobject

Interface for Person

```
interface DBObject {
    public int size();
    public String id();
    public void readInput();
    public void writeToFile(RandomAccessFile fw) throws IOException;
    public void readKey();
    public void readID(RandomAccessFile fr) throws IOException;
    public void readFromFile(RandomAccessFile fr) throws IOException;
    public void printRecord();
    final int IDSIZE = 10;
    final int NAMESIZE = 20;
    final int MAJORSIZE = 20;
}
```

Class Person

```
class Person implements DBObject{
    public String id, name, city;
    int year;
    double salary;
    String dbName = "Person.dat";
    int size = IDSIZE+NAMESIZE+NAMESIZE+4+8; //= 62 bytes
    Person(){      //where is the beginning of fifth person => 248 byte number
        id = "undefined"; name = "undefined"; city = "undefined";
        year =0; salary=0;
    }
    Person(String i, String n, String c,int y, double s){
        id = i; name = n; city =c;
        year=y; salary=s;
    }
```

Class Student

```
class Student extends Person {
    String major;
    String dbName = "Student.dat";
    int size = super.size + MAJORSIZE; // = 62 +20 =82 bytes
    Student(){
        super(); major = "Undefined";
    }
    Student(String i,String n,String c,int y,int s,String m){
        super(i,n,c,y,s); major=m;
    }
```

Class Database

```
class Database {
  RandomAccessFile dbFile;

  public void run(DBObject ob) throws IOException{
     int choice=1;
     Scanner in = new Scanner (System.in);
     System.out.println("Type 1 for add, 2 for search, 3 for edit, 0 for exit");
     while ((choice=in.nextInt())!=0){
        switch (choice){
        case 1: ob.readInput(); add(ob); break;
        case 2: ob.readKey();
              if (search(ob)) {
                ob.printRecord(); System.out.println();
              }
              break;
        }
        System.out.println("Type 1 for add, 2 for search, 3 for edit, 0 for exit");
     }
  }
}
```

Main

```
public static void main(String[] args) throws IOException{
    int choice=1;
    Scanner in = new Scanner(System.in);
    System.out.println("Type 1 for person, 2 for student, 0 for exit");
    while ((choice = in.nextInt()) != 0){
       switch (choice){
          case 0: break;
          case 1: (new Database()).run(new Person()); break;
          case 2:  (new Database()).run(new Student()); break;
          default: System.out.println("Invalid Choice."); break;
       }
       System.out.println("Type 1 for person, 2 for student, 0 for exit");
    }
  }
```

To add new Person

- In Person class

```
public void readInput() {
    Scanner in = new Scanner(System.in);
    System.out.println("What is the ID?"); id = in.nextLine();
    System.out.println("What is the name?"); name = in.nextLine();
    System.out.println("What is the the city of residence?"); city = in.nextLine();
    System.out.println("What is the year of birth?"); year = in.nextInt();
    System.out.println("What is the salary?"); salary = in.nextDouble();
  }
```

To add new Person

- In Database Class

```
private void add(DBObject ob) throws IOException{
      if (ob.getClass().getName() == "lecture4rafcasestudy.Person") {
         dbFile = new RandomAccessFile( new Person().dbName,"rw");
      }
      else if (ob.getClass().getName() == "lecture4rafcasestudy.Student"){
         dbFile = new RandomAccessFile( new Student().dbName,"rw");
      }
      dbFile.seek(dbFile.length()); //dbFile is a pointer. Point to the last
                  //position
      ob.writeToFile(dbFile);    // append at the end
      dbFile.close();
   }
```

To add new Person

- In Person class

```
public void writeToFile(RandomAccessFile fw) throws IOException{
    id += "            ";
    fw.writeBytes(id.substring(0,IDSIZE));
    name += "                    ";
    fw.writeBytes(name.substring(0,NAMESIZE));
    city += "                    ";
    fw.writeBytes(city.substring(0,NAMESIZE));
    fw.writeInt(year);
    fw.writeDouble(salary);
  }
```

To add new Student

- In Student class

```
public void readInput() {
        Scanner in = new Scanner(System.in);
        super.readInput();
        System.out.println("What is the major?");
        major = in.nextLine();
    }
    public void writeToFile(RandomAccessFile fw) throws IOException{
        super.writeToFile(fw);
        major += "                ";
        fw.writeBytes(major.substring(0,MAJORSIZE));
    }
```

To search Person

- In Person class

```
public void readKey() {
        Scanner in = new Scanner(System.in);
        System.out.println("What is the ID?");
        id = in.nextLine();
    }
```

To search Person

- In Database class

```
private boolean search(DBObject ob) throws IOException{
    if (ob.getClass().getName() == "lecture4rafcasestudy.Person") {
       dbFile = new RandomAccessFile( new Person().dbName,"r");
    }
    else if (ob.getClass().getName() == "lecture4rafcasestudy.Student"){
       dbFile = new RandomAccessFile( new Student().dbName,"r");
    }

    String tempID = ob.id();              dbFile.seek(0);
    while (dbFile.getFilePointer() < dbFile.length()){
       ob.readID(dbFile);
       if (ob.id().trim().equals((String) tempID)){
          ob.readFromFile(dbFile);      dbFile.close();          return true;
       }
       else
          dbFile.seek(dbFile.getFilePointer()+ob.size()- ob.IDSIZE);
    }
    System.out.println("Record not found.");
    dbFile.close();
    return false;
```

To search Person

- In Person class

```
public void readID(RandomAccessFile fr) throws IOException{
    byte [] tempid = new byte[IDSIZE];
    fr.read(tempid,0, IDSIZE);
    String tempid2 = new String(tempid);
    id = tempid2;
  }
```

To search Person

- In person class

```
public void readFromFile(RandomAccessFile fr) throws IOException{
        byte [] temp = new byte[NAMESIZE];
        fr.read(temp,0,NAMESIZE);
        String temp2 = new String(temp);
        name = temp2;
        fr.read(temp,0, NAMESIZE);
        temp2 = new String(temp);
        city = temp2;
        year = fr.readInt();
        salary = fr.readDouble();
    }
```

To search Person

- In Person class

```
public void printRecord(){
        System.out.print(id+" "+name+" "+city+"
                                    "+year+" "+salary);
    }
```

To search Student

- In Student Class

```
public void readFromFile(RandomAccessFile fr) throws IOException{
      super.readFromFile(fr);
      byte [] temp = new byte[10];
      fr.read(temp,0, MAJORSIZE);
      String temp2 = new String(temp);
      major = temp2;
   }
   public void printRecord(){
      super.printRecord();
      System.out.print(" "+major);
   }
```

Other Methods

- In Person class

```
public int size(){
    return size;
}
public String id(){
    return id.toString();
}
```

- In Student class