SEMINAR KĨ THUẬT PHẦN MỀM

MỘT SỐ VẤN ĐỀ CƠ BẢN VỀ JAVA

Nhóm 07 :

Nguyễn Hồng Phương
Phạm Thiên Phúc
Nguyễn Giáp Nguyên Sinh

Nội dung trình bày

1. Nguyên tắc hoạt động của Java. Khái niệm

Java platform:

+ Java SE: JRE

+ Java ME: MIDP

2. Công cụ môi trường phát triển (JDK, Eclipse).

3. Cú pháp Java (package, tên file, tên lớp, cách thừa kế lớp và thực thi giao diện)

4. Các loại Interfaces và lớp mảng Java

1-1. NGUYÊN TẮC HOẠT ĐỘNG

- Một chương trình viết bằng ngôn ngữ lập trình Java sẽ được biên dịch ra mã của máy ảo java (mã java bytecode). Sau đó máy ảo Java chịu trách nhiệm chuyển mã java bytecode thành mã máy tương ứng.

- Java là một ngôn ngữ lập trình vừa biên dịch vừa thông dịch. Chương trình nguồn viết bằng ngôn ngữ lập trình Java có đuôi *.java đầu tiên được biên dịch thành tập tin có đuôi *.class và sau đó sẽ được trình thông dịch thông dịch thành mã máy.

1-2. JAVA SE, JRE

- Java SE (Java Platform, Standard Edition): là công nghệ nền hỗ trợ xây dựng các ứng dụng có chức năng cao, tốc độ và đáng tin cậy

- JRE (Java Runtime environment): bao gồm máy ảo Java, các thư viện và các tập tin cần thiết. Là môi trường để thực thi một ứng dụng Java.

1-3. JAVA ME, MIDP

- Java ME (Java Platform, Micro Edition): platform để phát triển các ứng dụng trên thiết bị di động.

- MIDP (Mobile Information Device Profile): là môi trường để thực thi ứng dụng Java trên thiết bị di động.

2. CÔNG CỤ Và MÔI TRƯỜNG PHÁT TRIỂN JAVA

2-1. JDK – Java Development Kit

- Bộ công cụ phát triển java được cung cấp bởi sun microsystems. Mục đích của jdk là cung cấp phần mềm và các công cụ được yêu cầu cho việc biên dịch, kiểm tra lỗi và thực thi các chương trình java.
- JDK gồm 2 công cụ quan trọng
    - Javac (biên dịch)
    - Java (thông dịch)

2-1. JDK – JAVA DEVELOPMENT KIT

- Javac : được sử dụng để biên dịch mã nguồn của java sang dạng bytecode.
  - Cú pháp : javac [option] source

  Source là tập tin .java
- Java : được sử dụng để thông dịch và chạy các Java bytecode. Nó lấy tên của một tập tin class làm đối số để thực thi
  - Cú pháp : java [option] classname [arguments]

2-1. JDK – COMPILE VÀ RUN

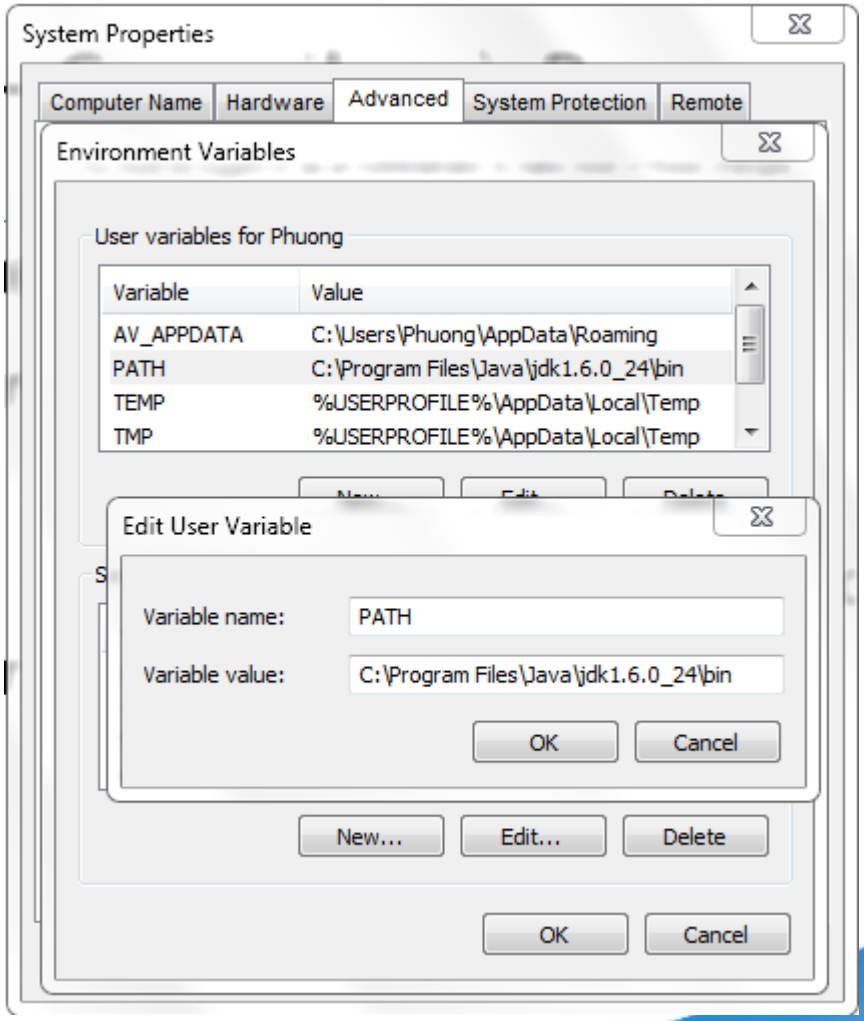
Bước 1: Download JDK

Bước 2: Cài đặt JDK

2-1. JDK – COMPILE VÀ RUN

Bước 3: Cấu hình máy (Win 7)

- Click phải vào Computer
- Properties
- Advanced system settings
- Chọn thẻ Advanced
- Environment Variables…



2-1. JDK – COMPILE VÀ RUN

Bước 3: (tiếp)

Ở phần user variables nếu:

+ Tồn tại variable PATH: chọn PATH và nhấn Edit > tại value thêm vào ;C:\Program Files\Java\jdk1.6.0_24\bin (đường dẫn đến thư mục bin nơi cài JDK)

+ Chưa tồn tại variable PATH: nhấn New > thêm variable name PATH và value C:\Program Files\Java\jdk1.6.0_24\bin

2-1. JDK – COMPILE VÀ RUN

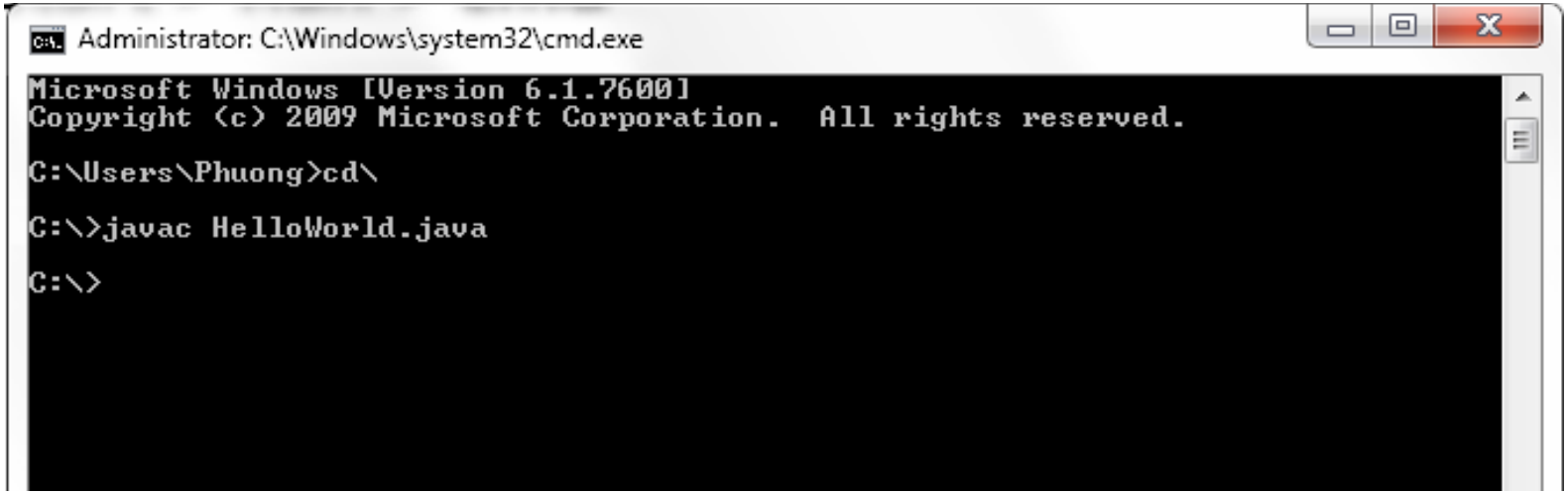
Bước 4: Mở Notepad và tạo file HelloWorld.java tại C:\ với nội dung:

```
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello World");
        }
}
```

2-1. JDK – COMPILE VÀ RUN

Bước 5: Biên dịch HelloWorld.java thành HelloWorld.class (dùng lệnh javac)

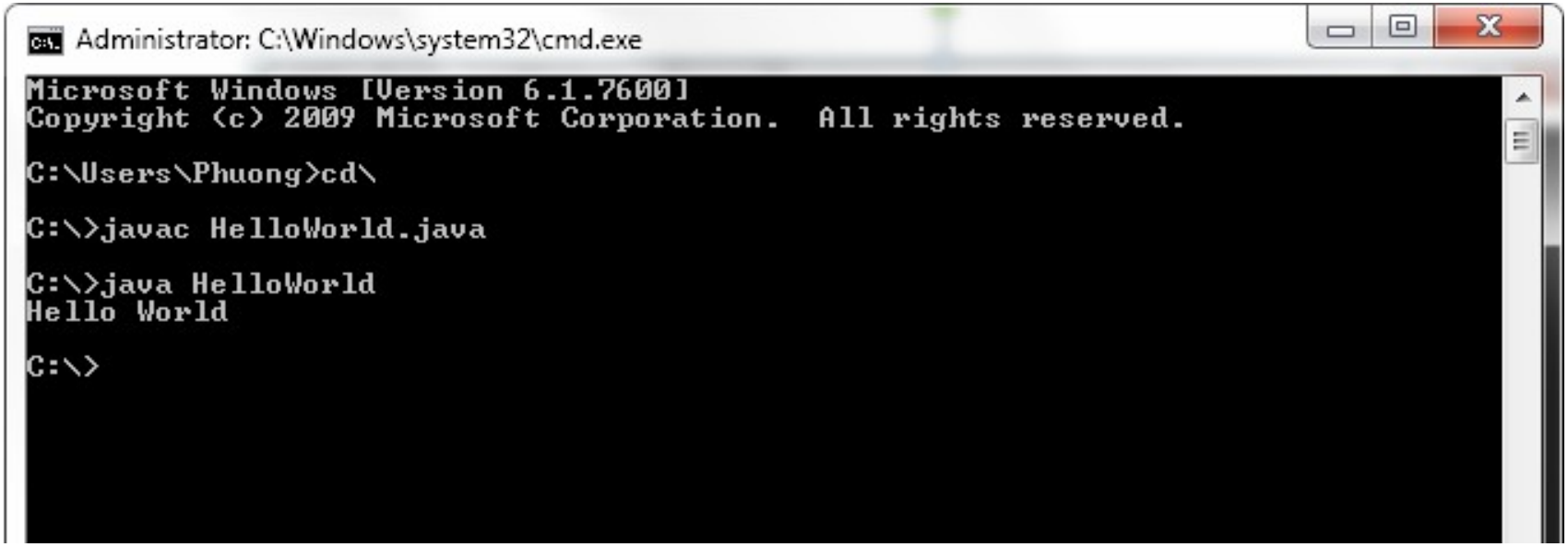
Start > Run > cmd > cd\ > javac HelloWorld.java



2-1. JDK – COMPILE VÀ RUN

Bước 6: Chạy (dùng lệnh java)

java HelloWorld > xem kết quả

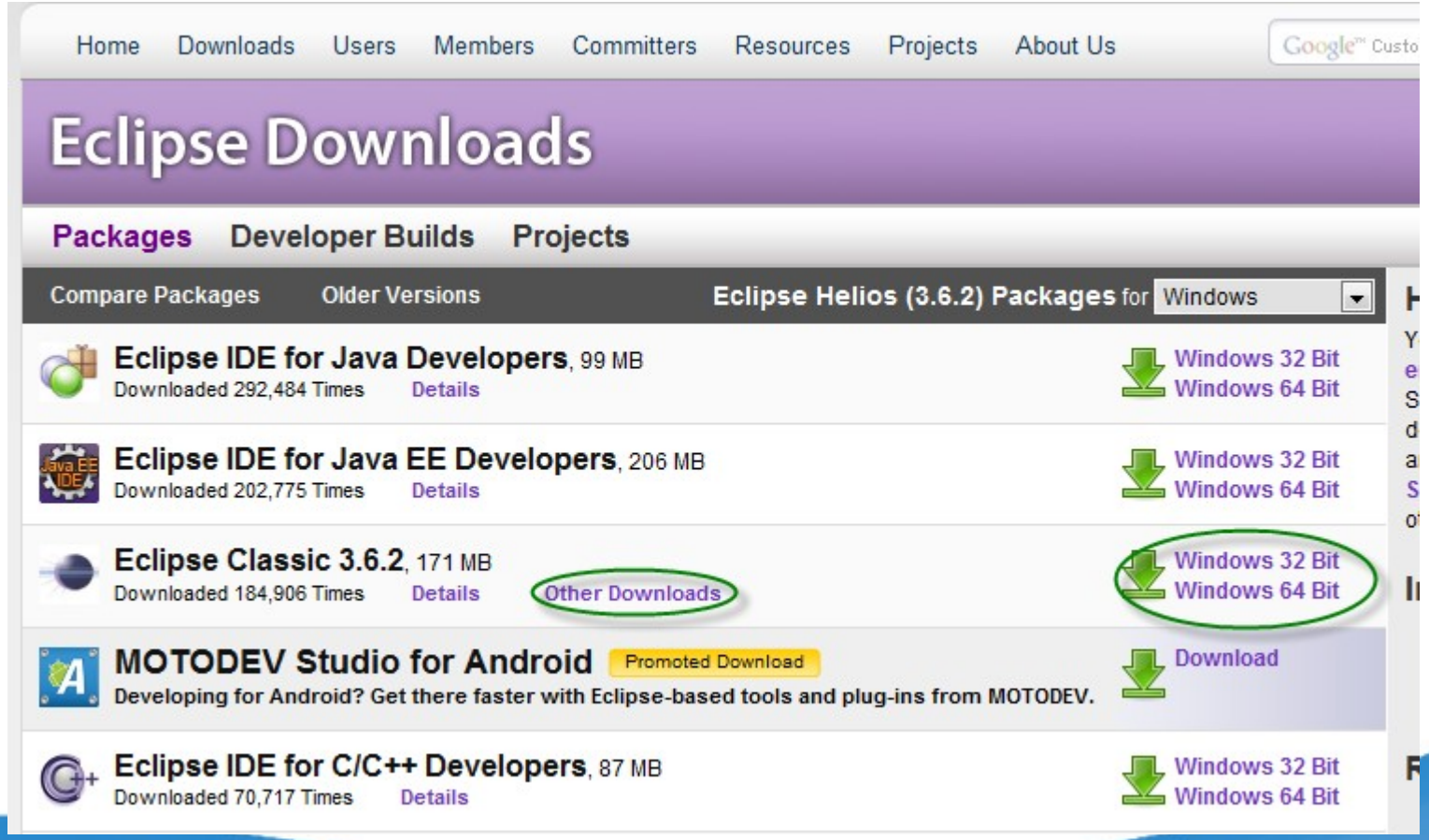


2-2. ECLIPSE – COMPILE VÀ RUN

- Eclipse là phần mềm miễn phí, được các nhà phát triển sử dụng để xây dựng những ứng dụng J2EE.
- Eclipse SDK bao gồm 3 phần chính:
  - **Platform**
  - **Java Development Toolkit (JDT)**
  - **Plug-in Development Environment (PDE)**.

2-2. ECLIPSE – COMPILE VÀ RUN

Bước 1: Download Eclipse



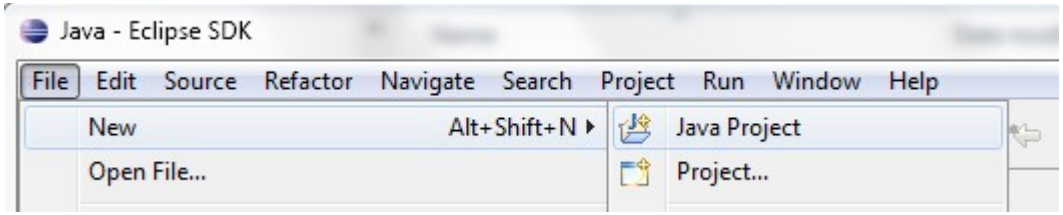
2-2. ECLIPSE – COMPILE VÀ RUN

Bước 2: Giải nén tập tin vừa tải

Bước 3: Tạo project Hello World

- Chạy eclipse.exe
- File > New > Java Project

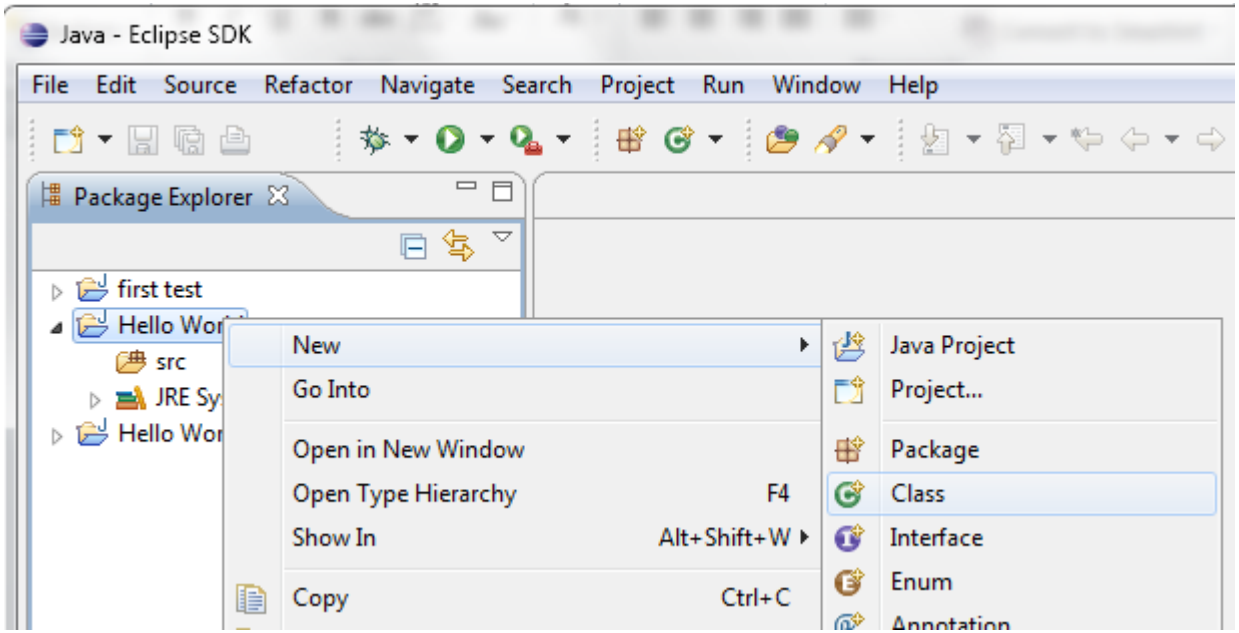
- Tai Proj



2-2. ECLIPSE – COMPILE VÀ RUN

Bước 4: Thêm class HelloWorld

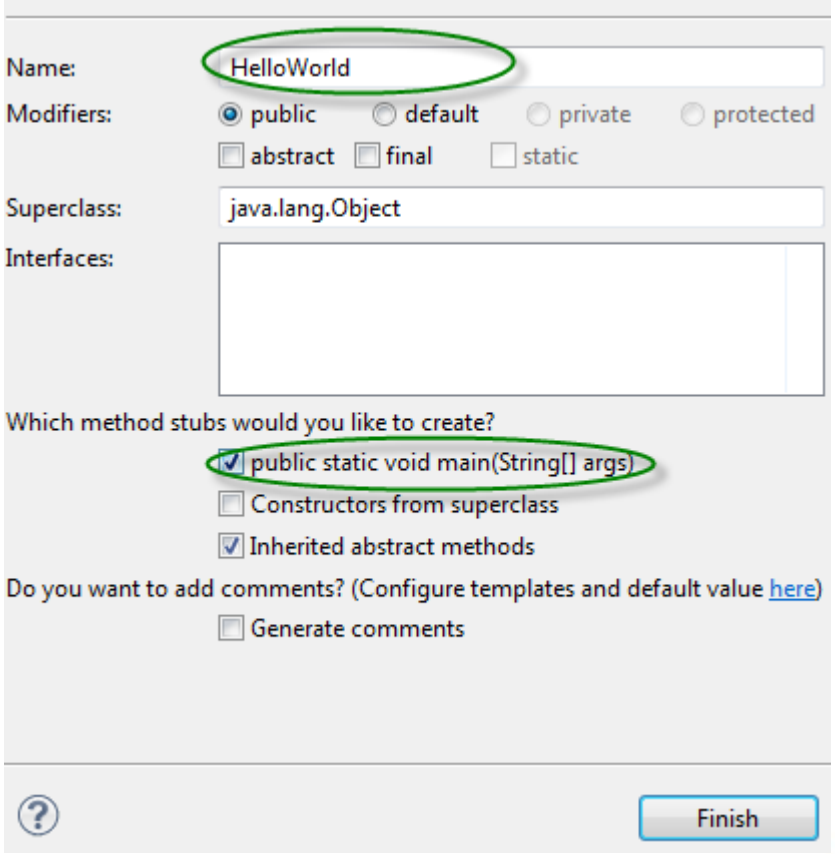
- Click phải vào Project Hello World > New > Class



2-2. ECLIPSE – COMPILE VÀ RUN

Bước 4: (tiếp)

- Đặt tên class là HelloWorld
- Check vào public static void main (String[] args)
- Click Finish

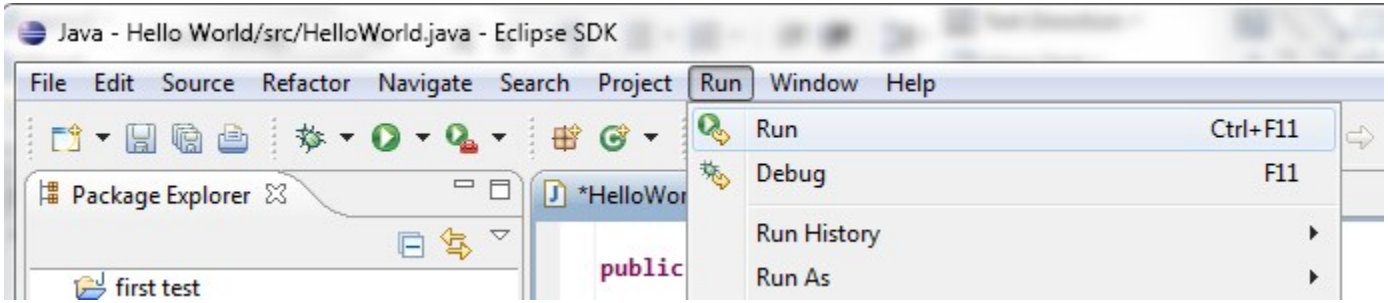


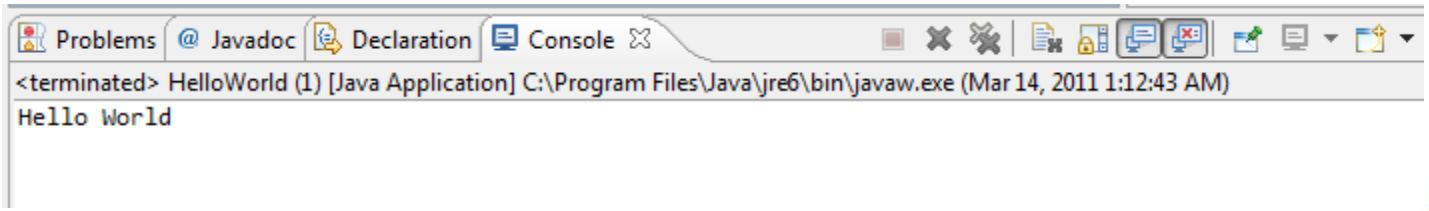
2-2. ECLIPSE – COMPILE VÀ RUN

Bước 5: Thêm vào hàm main đoạn code

System.out.println("Hello World");

Bước 6: Run và xem kết quả





2-3. NHẬN XÉT

- JDK:
  - Tạo một file text
  - Dùng javac để biên dịch file text thành file .class
  - Dùng java để thực thi file .class

- Eclipse: mọi thao tác thuận lợi hơn khi sử dụng giao diện để tương tác

3. CÚ PHÁP NNLT JAVA

3-1. Package

- Việc đóng gói các lớp lại tạo thành một thư viện dùng chung gọi là package.

- Một package có thể chứa một hay nhiều lớp bên trong, đồng thời cũng có thể chứa một package khác bên trong.

3-1. PACKAGE

-Để khai báo một lớp thuộc một gói nào đấy ta phải dùng từ khóa package.

-Dòng khai báo gói phải là dòng đầu tiên trong tập tin khai báo lớp.

- Các tập tin khai báo lớp trong cùng một gói phải được lưu trong cùng một thư mục.

3-1. PACKAGE

Ví dụ:

```
package phuongtiengiaothong;
class xemay
{
  // ….
}
```

Khi đó muốn sử dụng lớp *xemay* vào chương trình ta sẽ khai báo như sau:

```
import phuongtiengiaothong.xemay;
```

3-2. TÊN FILE, TÊN LỚP

- Tên lớp:
  + Bao gồm một chuỗi các ký tự (Unicode), ký số (Unicode), ký số.
  + Phải bắt đầu bằng một chữ cái, dấu gạch dưới ‘_’ hay dấu dollar '$'
  + Không được trùng với các từ khóa
  + Không có khoảng trắng ở giữa
- Tên file: file .java phải có tên trùng với tên lớp

3-3. KẾ THỪA

- Một lớp con (subclass) có thể kế thừa tất cả những vùng dữ liệu và phương thức của một lớp khác (siêu lớp - superclass).

- Như vậy việc tạo một lớp mới từ một lớp đã biết sao cho các thành phần (fields và methods) của lớp cũ cũng sẽ thành các thành phần (fields và methods) của lớp mới. Khi đó ta gọi lớp mới là lớp dẫn xuất (derived class) từ lớp cũ (superclass). Có thể lớp cũ cũng là lớp được dẫn xuất từ một lớp nào đó, nhưng đối với lớp mới vừa tạo thì lớp cũ đó là một lớp siêu lớp trực tiếp (immediate supperclass).

3-3. KẾ THỪA

Dùng từ khóa extends để chỉ lớp dẫn xuất.

```
class A extends B
{
 // …
}
```

3-4. Interface là gì?

- Theo phương pháp phân cấp thừa kế, có thể lớp cha có những hành vi chưa biết viết code thề nào → hành vi trừu tượng → lớp trừu tượng.
- Ta muốn một lớp được thừa kế từ nhiều lớp trừu tượng mà không bị khống chế bởi tính đơn thừa kế.
- Giải pháp: Interface

Interface...

- Interface là một khai báo bao gồm một tập đặc điểm gốm các hằng, các hành vi mà không muốn khai báo lớp.
- Interface mang ý nghĩa ***"như là một lớp hoàn toàn trừu tượng"***
- Interface mang ý nghĩa khai báo trước một nhóm các xử lý cần có.
- Interface là một đặc điểm của các ngôn ngữ OOP mới như Java, C#.
- Interface được xem như là một lớp ***hoàn toàn trừu tượng.***
- Interface là một công cụ để hiện thực dạng đa thừa kế trong Java, C#.

Khai báo interface

- Khai báo interface bằng 1 file.java, biên dịch thành file.class.
- Cú pháp

```
[modifier] interface InterfaceName
{  [ modifier] <final data>
   [ modifier] DataType Method (args);
}
```

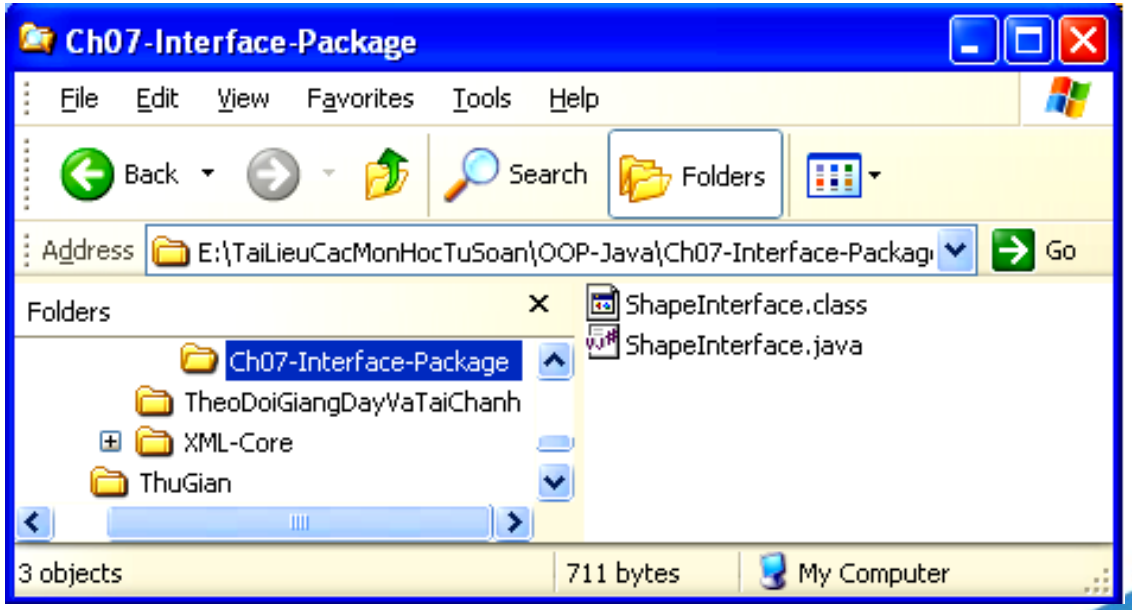
- Chọn modifier là public để mọi nơi đều dùng được.
- Không có modifier để chỉ cho cùng gói/ cùng thư mục truy cập.
- Modifier trong interface ***NÊN*** là public để dễ dùng.

Thí dụ:

ShapeInterface.java

```
1  // ShapeInterface.java
2  public interface ShapeInterface
3  { final public double Pi=3.141592;
4    final public double pi=3.141592;
5    public double calcArea();  // tinh dien tich
6    public double calcPerimeter(); // tinh chu vi
7  }
```

<u>Nhận xét:</u>
Các hình vẽ đều cần
-hằng PI
-tác vụ tính diện tích,
-tính chu vi
-mà ta không muốn tạo lớp trừu tượng
➔ Tạo 1 interface.

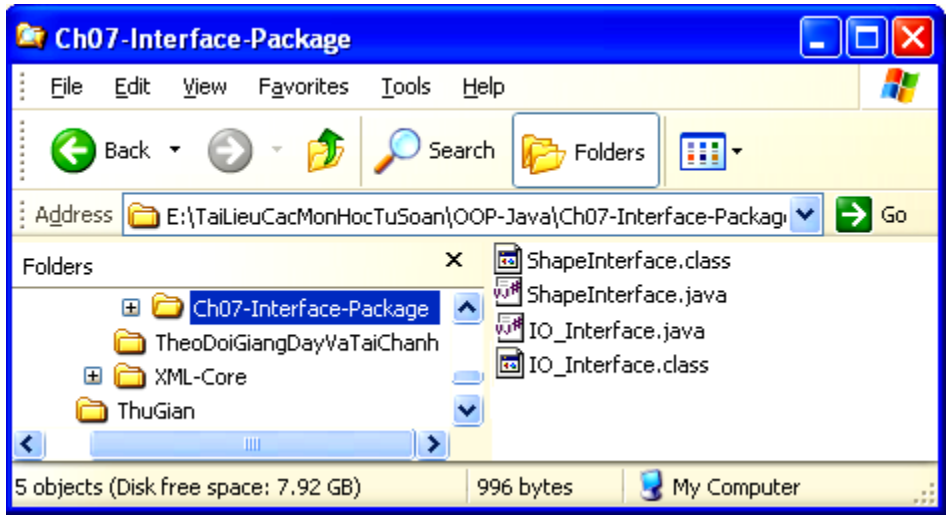


Thí dụ: interface về nhập xuất dữ liệu

- Ta nghĩ rằng, mỗi lớp trong phần mềm ta đang xây dựng đều cần nhập xuất dữ liệu. → Tạo một interface cho việc này.

IO_Interface.java

```
1  // IO_Interface.java
2  interface IO_Interface
3  { void input();
4    void output();
5  }
```



Hiện thực interface trong một lớp

- Một lớp có code cụ thể hóa các hành vi của một interface thì gọi là ***implementation.***
- Một lớp có thể hiện thực nhiều interface (có dạng đa thừa kế).
- Cú pháp xây dựng một lớp có hiện thực interface:

Cú pháp xây dựng lớp có hiện thực interface

[modifier] class Tên extends *LớpCha* implements *Interface1, interface2,...*
{

<Data riêng>

<hiện thực của các method riêng>

<hiện thực các method của interface1>

<hiện thực các method của interface2>

<hiện thực các method của interface khác>

}

Thứ tự không quan trọng

Thí dụ

InterfaceDemo_1.java

```
1  interface InterfaceDemo_1
2  { private void methodA();
3    protected void methodB();
4    void methodC();
5    public void methodD();
6  }
```

Task List

description
modifier private not allowed here
modifier protected not allowed here

Khai báo method trong interface phải là *friendly* hoặc public.
Suy nghĩ về đặc điểm: *interface là quy định về các khả năng của một lớp.*

InterfaceDemo_1.java

```
1  interface InterfaceDemo_1
2  { void methodA();
3  }
4  class C1 implements InterfaceDemo_1
5  {
6  }
```

Task List

description
C1 is not abstract and does not override abstract method methodA() in InterfaceDemo_1

Một lớp cụ thể có khai báo inplements một interface mà *quên chưa hiện thực hành vi của interface → Error*

Thí dụ:

InterfaceDemo_1.java *

```
1  interface InterfaceDemo_1
2  { void methodA();
3    void methodB();
4  }
5  abstract class C1 implements InterfaceDemo_1
6  {  abstract public void methodC();
7  }
8  class C2 extends C1
9  { void methodA() { System.out.print("method A");}
10   void methodB() { System.out.print("method B");}
11   void methodC() { System.out.print("method C");}
12 }
13
```

Lớp trừu tượng có thể chưa cụ thể các method của interface

Modifier của hành vi cụ thể của các method trong interface phải là public

Task List

description
methodC() in C2 cannot override methodC() in C1; attempting to assign weaker access privileges; was public
methodB() in C2 cannot implement methodB() in InterfaceDemo_1; attempting to assign weaker access privileges; was public
methodA() in C2 cannot implement methodA() in InterfaceDemo_1; attempting to assign weaker access privileges; was public

Thí dụ: Biến interface

InterfaceDemo_1.java

```
1  interface InterfaceDemo_1
2  { void methodA();
3    void methodB();
4  }
5  abstract class C1 implements InterfaceDemo_1
6  {  abstract public void methodC();
7  }
8  class C2 extends C1
9  { public void methodA() { System.out.println("method A");}
10   public void methodB() { System.out.println("method B");}
11   public void methodC() { System.out.println("method C");}
12 }
13 class C2Demo
14 {  public static void main (String args[])
15    {  C2 obj1= new C2();
16       obj1.methodA();
17       C1 obj2 = new C2();
18       obj2.methodB();
19       InterfaceDemo_1 obj3= new C2();
20       obj3.methodB();
21       obj3.methodC();
22    }
23 }
```

Khai báo biến thông qua interface

Biến interface chỉ được dùng với các hành vi có trong interface

Task List

description
cannot find symbol method methodC()

Thí dụ: 2 interface cụ thể

interface cho việc tính toán các hình vẽ

ShapeInterface.java | IO_Interface.java | VongTron.java

```
1  // ShapeInterface.java
2  public interface ShapeInterface
3  { final public double Pi=3.141592;
4    final public double pi=3.141592;
5    public double calcArea();  // tinh dien tich
6    public double calcPerimeter(); // tinh chu vi
7  }
```

interface cho việc nhập xuất đối tượng

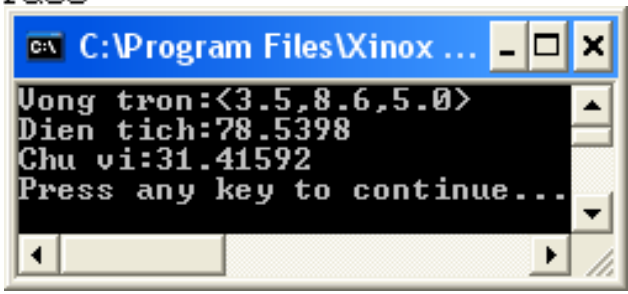
ShapeInterface.java | IO_Interface.java | VongTron.java

```
1  // IO_Interface.java
2   interface IO_Interface
3  { void input();  // nhap object
4    void output(); // xuat object
5  }
```

Thí dụ: Sử dụng 2 interface

ShapeInterface.java | IO_Interface.java | **VongTron.java**

```
1  // VongTron.java
2  import javax.swing.*;
3  class VongTron implements ShapeInterface,  IO_Interface
4  {  protected double x,y,r;
5     VongTron() { x=y=r=0;}
6     VongTron(double x, double y, double r)
7        { this.x=x; this.y=y; this.r=r;
8        }
9     public double calcArea() { return Pi*r*r;}
10    public double calcPerimeter() { return 2*Pi*r;}
11    public void input()
12    { String S=JOptionPane.showInputDialog(null,"Toa do x:","0");
13      x= Double.parseDouble(S);
14      S=JOptionPane.showInputDialog(null,"Toa do y:","0");
15      y= Double.parseDouble(S);
16      S=JOptionPane.showInputDialog(null,"Ban kinh:","0");
17      r= Double.parseDouble(S);
18    }
19    public void output()
20    {  System.out.println("Vong tron:<" + x + "," + y + "," + r + ">");
21    }
22 }
23
24 class VongTronDemo
25 {  public static void main(String srgs[])
26     {  VongTron  v= new VongTron();
27        v.input();
28        v.output();
29        System.out.println("Dien tich:" + v.calcArea());
30        System.out.println("Chu vi:" + v.calcPerimeter());
31     }
32 }
```



Interface thừa kế

- Interface có thể thừa kế interface cha
- Interface con là hội lại các khai báo hành vi.
- Cú pháp:

```
interface InterfaceSon extends InterfaceFather
{  < các khai báo thêm>
}
```

- Nếu một lớp implements interface con thì phải cụ thể hóa cả các method có trong interface cha.

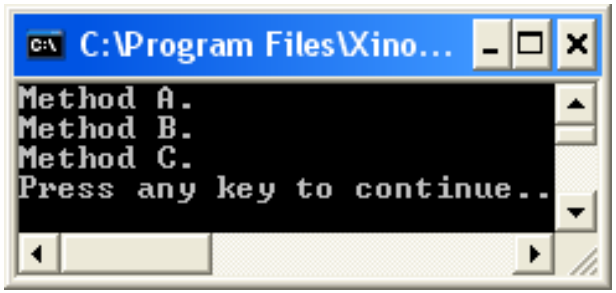
Thí dụ

InterfaceChaCon.java

```
1  // InterfaceChaCon.java
2
3  interface I_Cha
4  { void methodA();
5    void methodB();
6  }
7  interface I_Con extends I_Cha
8  { void methodC();
9  }
10 class A implements I_Con
11 { public void methodC()
12     { System.out.println("Method C.");
13     }
14 }
15
```

Task List

	description
✖	A is not abstract and does not override abstract method methodB() in I_Cha



InterfaceChaCon.java

```
1  // InterfaceChaCon.java
2
3  interface I_Cha
4  { void methodA();
5    void methodB();
6  }
7  interface I_Con extends I_Cha
8  { void methodC();
9  }
10 class A implements I_Con
11 { public void methodC()
12     { System.out.println("Method C.");
13     }
14   public void methodA()
15     { System.out.println("Method A.");
16     }
17   public void methodB()
18     { System.out.println("Method B.");
19     }
20 }
21 class I_Cha_Con_Demo
22 {  public static void main(String args[])
23     { A obj= new A();
24       obj.methodA();
25       obj.methodB();
26       obj.methodC();
27     }
28 }
```

Collections Framework

- Collections Framework bao gồm nhiều lớp và giao diện trong khung cộng tác.

- Khung cộng tác của các Collection Java dựa trên triển khai thực hiện cụ thể một số giao diện định nghĩa các kiểu sưu tập (collection)

Collection

- **Collection** là đối tượng có khả năng chứa các đối tượng khác.
- Các thao tác thông thường trên collection
  - Thêm/Xoá đối tượng vào/khỏi collection
  - Kiểm tra một đối tượng có ở trong collection không
  - Lấy một đối tượng từ collection
  - Duyệt các đối tượng trong collection
  - Xoá toàn bộ collection

Collections Framework

- Các collection đầu tiên của Java:
  - Mảng
  - Vector: Mảng động
  - Hastable: Bảng băm
- Collections Framework (từ Java 1.2)
  - Là một kiến trúc hợp nhất để biểu diễn và thao tác trên các collection.
  - Giúp cho việc xử lý các collection độc lập với biểu diễn chi tiết bên trong của chúng.

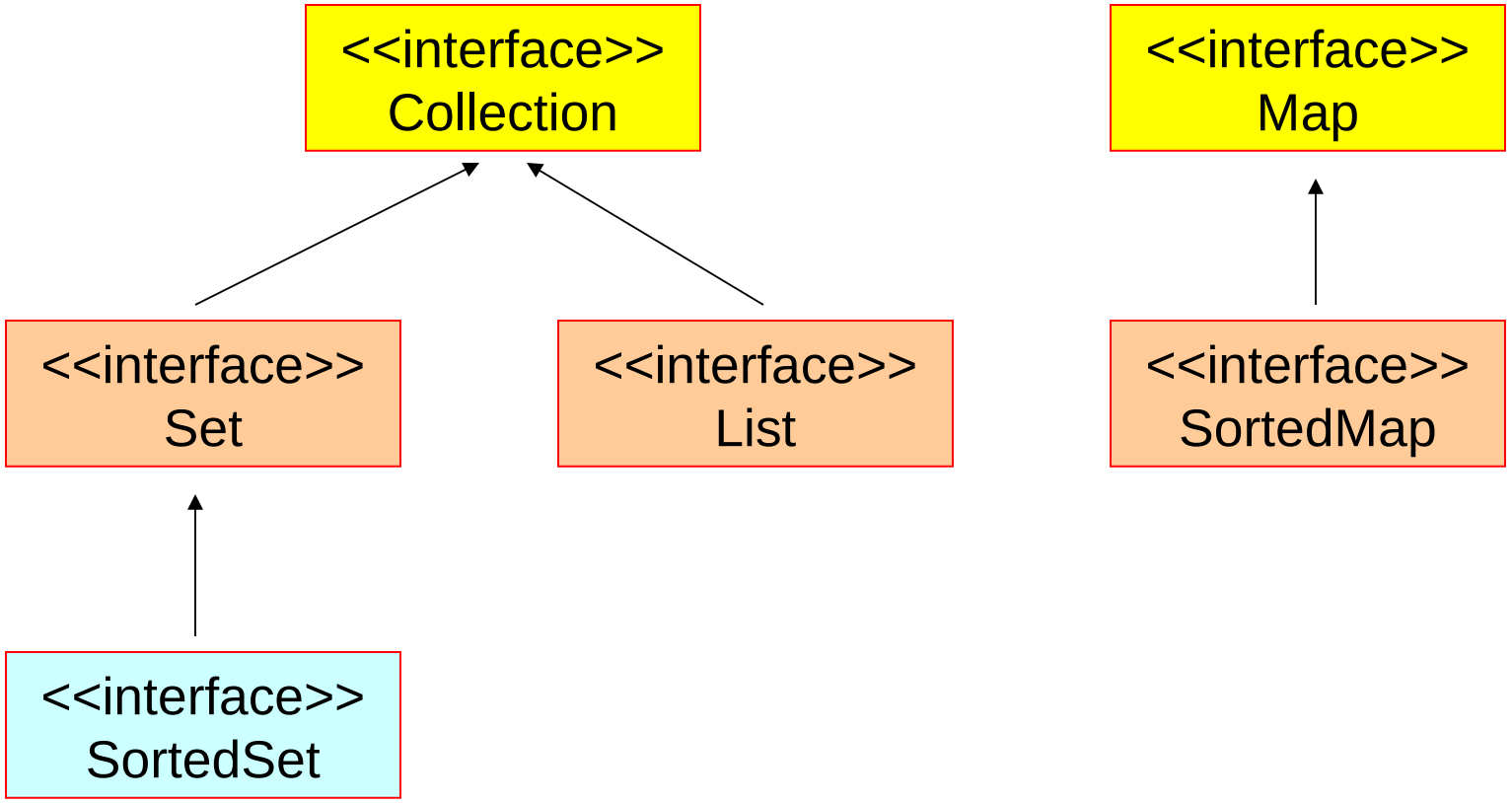
Collections Framework

- Một số lợi ích của Collections Framework
  - Giảm thời gian lập trình
  - Tăng cường hiệu năng chương trình
  - Dễ mở rộng các collection mới
  - Khuyến khích việc sử dụng lại mã chương trình

Collections Framework

- Collections Framework bao gồm
  - **Interfaces:** Là các giao tiếp thể hiện tính chất của các kiểu collection khác nhau như **List, Set, Map.**
  - **Implementations:** Là các lớp collection có sẵn được cài đặt các collection interfaces.
  - **Algorithms:** Là các phương thức tĩnh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...

Interfaces


<<interface>>
Collection
<<interface>>
Map
<<interface>>
Set
<<interface>>
List
<<interface>>
SortedMap
<<interface>>
SortedSet

Giao tiếp Collection

- Cung cấp các thao tác chính trên collection như thêm/xoá/tìm phần tử... Ví dụ:
  - **boolean add(Object element);**
  - **boolean remove(Object element);**
  - **boolean contains(Object element);**
  - **int size();**
  - **boolean isEmpty();**
- Nếu lớp cài đặt Collection không muốn hỗ trợ các thao tác làm thay đổi collection như add, remove, clear... nó có thể tung ra ngoại lệ UnsupportedOperationException.

Giao tiếp Set

- **Set** kế thừa từ Collection, hỗ trợ các thao tác xử lý trên collection kiểu tập hợp

- Set không có thêm phương thức riêng ngoài các phương thức kế thừa từ Collection.

Giao Tiếp **Set** định nghĩa bộ Collection không có phần tử trùng lắp

Giao tiếp SortedSet

- **SortedSet** kế thừa từ Set, nó hỗ trợ thao tác trên tập hợp các phần tử có thể so sánh được. Các đối tượng đưa vào trong một **SortedSet** phải cài đặt giao tiếp Comparable hoặc lớp cài đặt **SortedSet** phải nhận một Comparator trên kiểu của đối tượng đó.

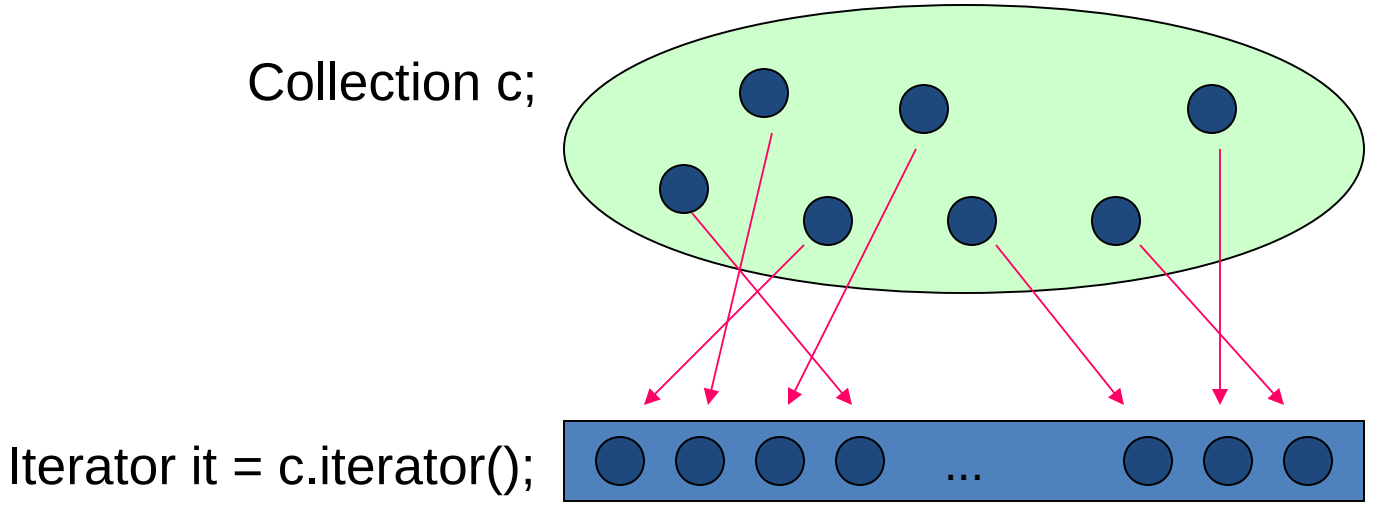
- Một số phương thức của **SortedSet**:
  - **Object first(); // lấy phần tử đầu tiên (nhỏ nhất)**
  - **Object last(); // lấy phần tử cuối cùng (lớn nhất)**
  - **SortedSet subSet(Object e1, Object e2); // lấy một tập các phần tử nằm trong khoảng từ e1 tới e2.**

Giao tiếp List

- List kế thừa từ Collection, nó cung cấp thêm các phương thức để xử lý collection kiểu danh sách (Danh sách là một collection với các phần tử Object được xếp theo chỉ số).
- Một số phương thức của List
  - **Object get(int index);**
  - **Object set(int index, Object o);**
  - **void add(int index, Object o);**
  - **Object remove(int index);**
  - **int indexOf(Object o);**
  - **int lastIndexOf(Object o);**

Duyệt collection

- Các phần tử trong collection có thể được duyệt thông qua Iterator.
- Các lớp cài đặt Collection cung cấp phương thức trả về iterator trên các phần tử của chúng.



Duyệt collection

- **Iterator** cho phép duyệt tuần tự một collection.
- Các phương thức của Iterator:
  - **boolean hasNext();**
  - **Object next();**
  - **void remove();**
- Ví dụ:

```
Iterator it = c.iterator();
while ( it.hasNext() ) {
    Point p = (Point) it.next();
    System.out.println( p.toString() );
}
```

Giao tiếp Map

- Giao tiếp **Map** cung cấp các thao tác xử lý trên các bảng ánh xạ. định nghĩa collection có các cặp khóa - giá trị.
- Một số phương thức của Map
  - **Object put(Object key, Object value);**
  - **Object get(Object key);**
  - **Object remove(Object key);**
  - **boolean containsKey(Object key);**
  - **boolean containsValue(Object value);**
  - **...**

Giao tiếp Map

- Map cung cấp 3 cách view dữ liệu:
  - View các khoá:

    Set keySet(); // Trả về các khoá
  - View các giá trị:

    Collection values(); // Trả về các giá trị
  - View các cặp khoá-giá trị

    Set entrySet(); // Trả về các cặp khoá-giá trị
- Sau khi nhận được kết quả là một collection, ta có thể dùng iterator để duyệt các phần tử của nó.

Giao tiếp SortedMap

- Giao tiếp SortedMap kế thừa từ Map, nó cung cấp thao tác trên các bảng ánh xạ với khoá có thể so sánh được.

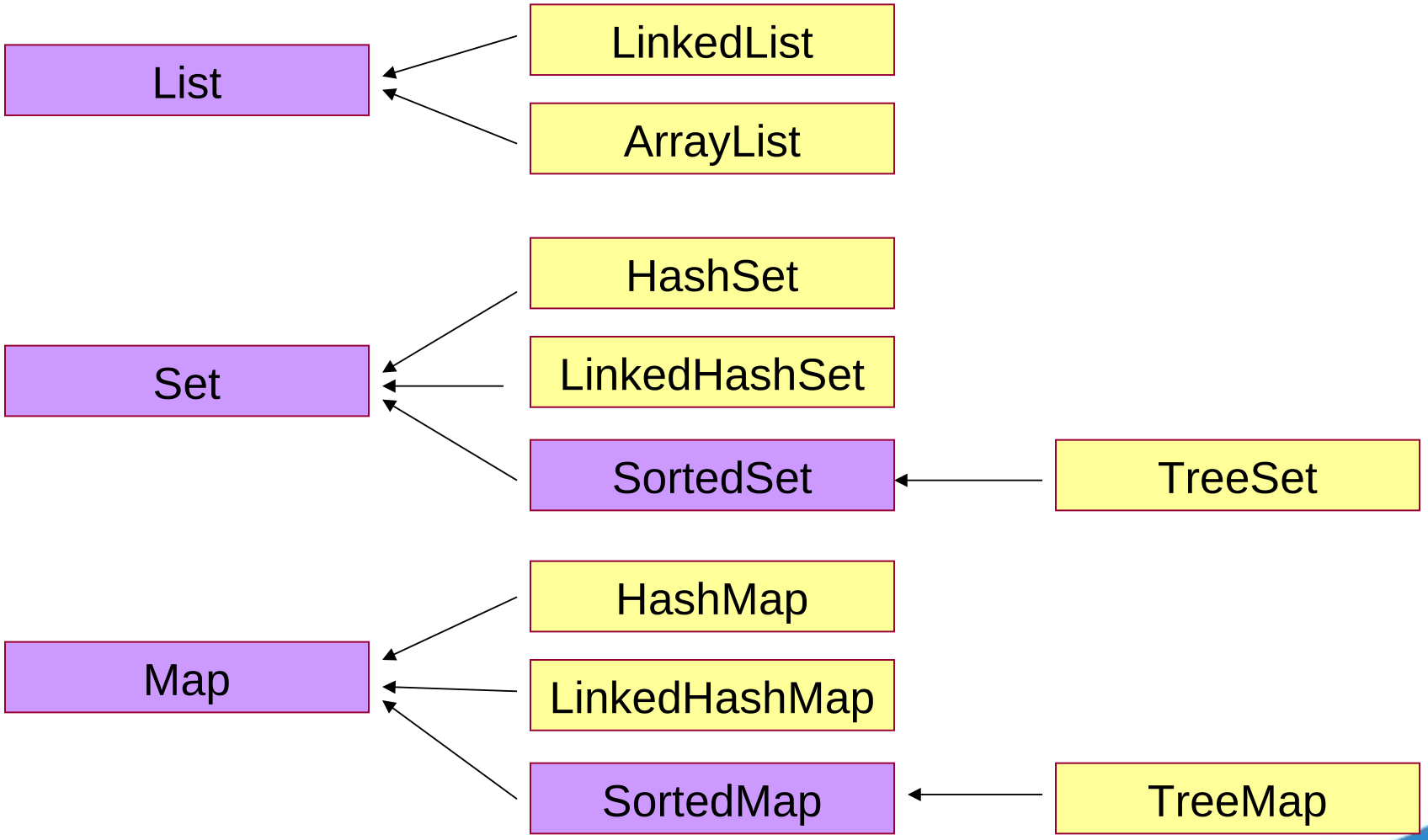
- Giống như SortedSet, các đối tượng khoá đưa vào trong SortedMap phải cài đặt giao tiếp Comparable hoặc lớp cài đặt SortedMap phải nhận một Comparator trên đối tượng khoá.

Implementations

- Các cài đặt trong **Collections Framework** chính là các lớp collection có sẵn trong Java.

  Chúng cài đặt các **collection interface** ở trên để thể hiện các cấu trúc dữ liệu cụ thể. Ví dụ: mảng động, danh sách liên kết, cây đỏ đen, bảng băm...

Implementations



Mô tả các cài đặt

- ArrayList: Mảng động, nếu các phần tử thêm vào vượt quá kích cỡ mảng, mảng sẽ tự động tăng kích cỡ.
- LinkedList: Danh sách liên kết 2 chiều. Hỗ trợ thao tác trên đầu và cuối danh sách.
- HashSet: Bảng băm.
- LinkedHashSet: Bảng băm kết hợp với linked list nhằm đảm bảo thứ tự các phần tử.
- TreeSet: Cây đỏ đen (red-black tree).

Mô tả các cài đặt

- HashMap: Bảng băm (cài đặt của Map).
- LinkedHashMap: Bảng băm kết hợp với linked list nhằm đảm bảo thứ tự các phần tử (cài đặt của Map).
- TreeMap: Cây đỏ đen (cài đặt của Map).

Ví dụ 1: TreeSet

```
// This program sorts a set of names
import java.util.*;

public class TreeSetTest1
{
    public static void main(String[] args)
    {
        SortedSet names = new TreeSet();
        names.add(new String("Minh Tuan"));
        names.add(new String("Hai Nam"));
        names.add(new String("Anh Ngoc"));
        names.add(new String("Trung Kien"));
        names.add(new String("Quynh Chi"));
        names.add(new String("Thu Hang"));
        System.out.println(names);
    }
}
```

Ví dụ 2: Student Set

```
class Student implements Comparable
{
    private String code;
    private double score;

    public Student(String code, double score)
    {
        this.code = code;
        this.score = score;
    }

    public double getScore()
    {
        return score;
    }

    public String toString()
    {
        return "(" + code + "," + score + ")";
    }
```

Ví dụ 2: Student Set

```
    public boolean equals(Object other)
    {
        Student otherStu = (Student) other;
        return code.equals(otherStu.code);
    }

    public int compareTo(Object other)
    {
        Student otherStu = (Student) other;
        return code.compareTo(otherStu.code);
    }
}
```

Ví dụ 2: Student Set

```
// This programs sorts a set of students by name and then by score
import java.util.*;

public class TreeSetTest2
{
    public static void main(String[] args)
    {
        SortedSet stu = new TreeSet();
        stu.add(new Student("A05726", 8.5));
        stu.add(new Student("A06338", 7.0));
        stu.add(new Student("A05379", 7.5));
        stu.add(new Student("A06178", 9.5));

        System.out.println(stu);

        SortedSet sortByScore = new TreeSet(new Comparator()
            // create an inner class
```

Ví dụ 2: Student Set

```
            {
                public int compare(Object a, Object b)
                {
                    Student itemA = (Student) a;
                    Student itemB = (Student) b;
                    double scoreA = itemA.getScore();
                    double scoreB = itemB.getScore();
                      if ( scoreA < scoreB )
                            return -1;
                        else
                            return 1;
                }
            }); // end of inner class

        sortByScore.addAll(stu);
        System.out.println(sortByScore);
    }
}
```

Ví dụ 3: HashMap

```
// This program stores a phone directory by hashing
import java.util.*;

public class MyMapTest
{
    public static void main(String[] args)
    {
        Map phoneDir = new HashMap();
        phoneDir.put("5581814", new String("Dept. Informatics"));
        phoneDir.put("8584490", new String("Defense Staff"));
        phoneDir.put("8587346", new String("Administrative Staff"));
        phoneDir.put("7290028", new String("Student Club"));

        // print all entries
        System.out.println(phoneDir);

        // remove an entry
        phoneDir.remove("8584490");
```

Ví dụ 3: HashMap

```
        // replace an entry
        phoneDir.put("7290028", new String("International Relation"));

        // look up a value
        System.out.println(phoneDir.get("5581814"));

        // iterate through all entries
        Set entries = phoneDir.entrySet();
        Iterator iter = entries.iterator();
        while (iter.hasNext())
        {
            Map.Entry entry = (Map.Entry) iter.next();
            String key = (String) entry.getKey();
            String value = (String) entry.getValue();
            System.out.println("key=" + key + ", value=" + value);
        }
    }
}
```

Các lớp bao

- Collection chỉ làm việc trên các Object. Những kiểu dữ liệu cơ bản như: byte, short, int, long, double, float, char, boolean không thể đưa được trực tiếp vào Collection mà phải thông qua các lớp bao.
- Các lớp bao: Byte, Short, Int, Long, Double, Float, Char, Boolean.
- Ví dụ:
  - Integer intObject = new Integer(9);
  - int value = intObject.intValue();

Algorithms

- Các thuật toán được cài đặt như những phương thức tĩnh của lớp Collections
- Một số phương thức của Collections:
  - static Object max(Collection c)
  - static Object min(Collection c)
  - static int binarySearch(List list, Object key)
  - static void sort(List list)
  - static void shuffle(List list)
  - các phương thức tạo synchronized collection
  - các phương thức tạo read-only collection

Ví dụ: Trộn bài

```
import java.util.*;

public class MyShuffleTest
{
    public static void main(String[] args)
    {
        List numbers = new ArrayList(52);

        for (int i = 1; i <= 52; i++)
            numbers.add(new Integer(i));

        System.out.println("Before shuffling:" + numbers + "\n");

        Collections.shuffle(numbers);
        System.out.println("After shuffling:" + numbers + "\n");
    }
}
```

Collections Framework

- **Legacy Implementations**
  - Là các lớp cũ được cài đặt bổ sung thêm các collection interface.
  - Vector: Có thể thay bằng ArrayList
  - Hastable: Có thể thay bằng HashMap
- **Abstract Implementations**
  - Là các lớp trừu tượng đã cài đặt các collection interface mà ta có thể kế thừa để tạo ra các collection mới.
  - AbstractCollection, AbstractSet, AbstractList...