LẬP TRÌNH JAVA

Bài 7:
LẬP TRÌNH VỚI CẤU TRÚC COLLECTION

ÔN TẬP

1. 2 loại error: syntax error, run-time error.
2. run-time error được gọi là exception.
3. Khi có một exception, JVM xuất động một object (chứa mô tả về ngoại lệ).
4. Lớp trên cùng của các error là lớp Throwable (lớp cho các đối tượng xuất động của JVM)
5. Có rất nhiều Exception để trong các gói thư viện của Java.

ÔN TẬP

6. Bẫy lỗi bằng try ... catch... finally
7. Nếu hành vi chứa mã nguồn có thể gây lỗi, chỉ thị bằng **throws** sau nguyên mẫu hàm.
8. Trong mã nguồn của hành vi, có thể xuất một exception bằng ***throw new ExceptionClass ("Msg");***
9. Tự định nghĩa một Exception là khai báo một lớp kế thừa một Exception đã có.

LẬP TRÌNH JAVA

ÔN TẬP

10. Java cung cấp sẵn một garbage collector.
11. Chủ động gọi GC bằng System.gc() hoặc thông qua một đối tượng RunTime.
12. class RunTime chứa thông tin về môi trường thức thi Java app.
13. Thời gian trong Java là một số long theo đơn vị mili giây kể từ 1-1-1970.
14. Sử dụng java.util.Date để thao tác với dữ liệu thời gian.
15. Có thể thông qua đối tượng System để lấy thời gian hiện hành của máy (theo mili, nano second).

BÀI 7: LẬP TRÌNH VỚI CẤU TRÚC COLLECTION

- Tuyển tập (giáo trình trang 181 đến 199)
  - Ý nghĩa của tuyển tập
  - Cấu tạo tuyền tập
  - Các thao tác trên tuyển tập
- Lớp tiện ích Collections (giáo trình trang 199 đến 210)
  - Tìm max,tìm min
  - Sắp xếp các phần tử
- Duyệt tuyển tập (giáo trình trang 188)
  - Iterator

1. TUYỂN TẬP - COLLECTIONS FRAMEWORK

- Ý nghĩa của tuyển tập.
  - Các cấu trúc tuyển tập cho phép tập hợp các đối tượng lại để xử lý như một đơn vị.
  - Các đối tượng bất kỳ có thể được lưu trữ, tìm kiếm và được thao tác như các phần tử của tuyển tập.
  - Cấu trúc tuyển tập có các **lớp tiện ích** phục vụ cho việc xử lý trên tuyển tập

1. TUYỂN TẬP- COLLECTIONS FRAMEWORK

- Các thao tác trên tuyển tập
  - Thêm/Xoá đối tượng vào/khỏi collection
  - Kiểm tra sự tồn tại của đối tượng trong collection
  - Lấy một đối tượng từ collection
  - Duyệt các đối tượng trong collection
  - Xoá toàn bộ collection
- Hai hệ thống phân cấp dẫn đầu là: Collection và Map

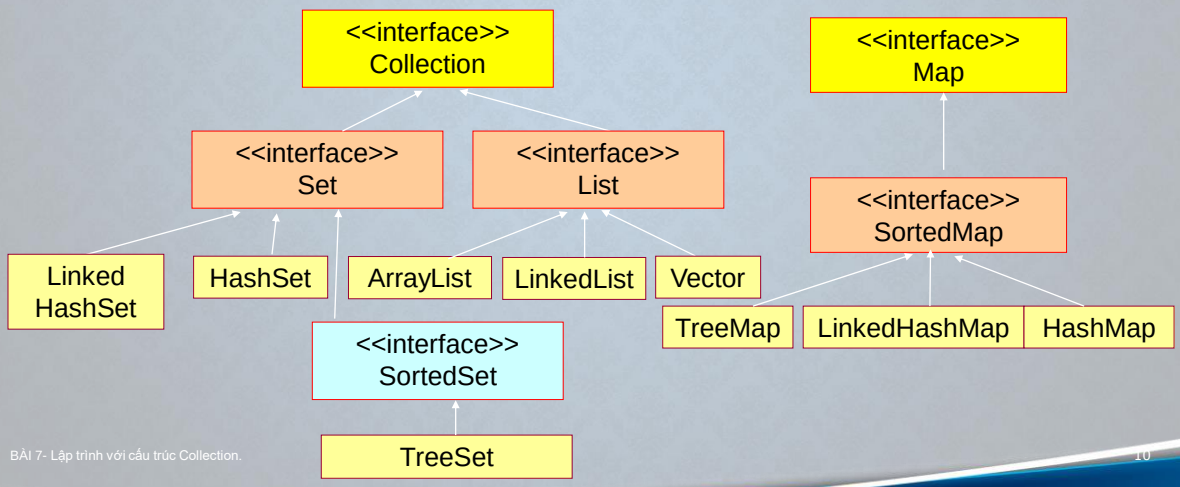
2. TUYỂN TẬP- COLLECTIONS FRAMEWORK

- 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

2. TUYỂN TẬP- COLLECTIONS FRAMEWORK

- Cấu tạo tuyển tập
  - 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.

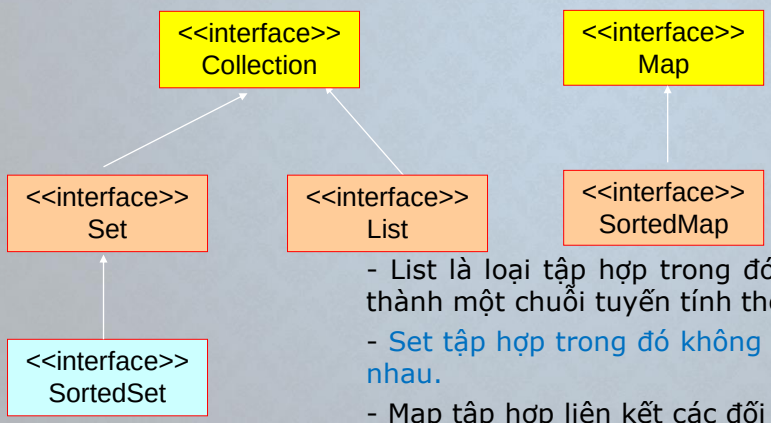
2. TUYỂN TẬP- SƠ ĐỒ - COLLECTIONS FRAMEWORK



2. COLLECTIONS FRAMEWORK (TT)

- Cấu tạo tuyển tập (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...

2.1. INTERFACES



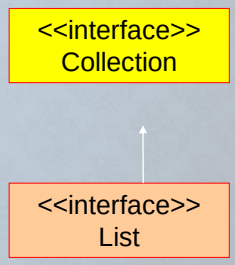
- List là loại tập hợp trong đó các đối tượng được sắp xếp thành một chuỗi tuyến tính theo chỉ số.
- Set tập hợp trong đó không cho phép các đối tượng trùng nhau.
- Map tập hợp liên kết các đối tượng thuộc một tập hợp này với 1 tập hợp khác như từ điển Map(T,S).

2.1. INTERFACES (TT)

- 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 thực hiện các phương thức theo nguyên mẫu đã định nghĩa, thì các phương thức đó có thể tạo ra ngoại lệ UnsupportedOperationException.

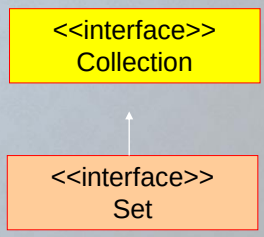
2.1. (TT). GIAO TIẾP LIST

- List kế thừa từ Collection- List cung cấp thêm một số phương thức để xử lý collection kiểu danh sách (xếp theo chỉ số).
- Một số phương thức của List
  - Object get(int index);
  - Object set(int index, Object o);//sửa dữ liệu
  - void add(int index, Object o); //insert tại index
  - Object remove(int index);
  - int indexOf(Object o);
  - int lastIndexOf(Object o);



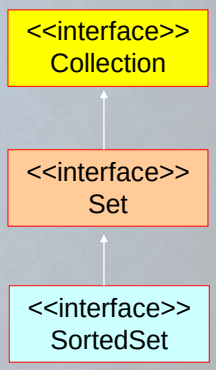
2.1. (TT). GIAO TIẾP SET

- Set kế thừa từ Collection, hỗ trợ các thao tác xử lý trên collection kiểu tập hợp (các phần tử không được trùng lặp).
- Set không có thêm phương thức riêng ngoài các phương thức kế thừa từ Collection.
  - boolean add(Object element);
  - boolean remove(Object element);
  - boolean contains(Object element);
  - int size();
  - boolean isEmpty();



2.1. (TT). GIAO TIẾP SORTEDSET

- SortedSet kế thừa từ Set hỗ trợ thao tác so sánh các phần tử.
- 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 (trang bên)



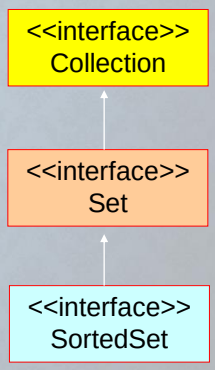
2.1. (TT). GIAO TIẾP SORTEDSET

- Một số phương thức của SortedSet:
  - Object first();

  // lấy phần tử đầu tiên (nhỏ nhất)
  - Object last();

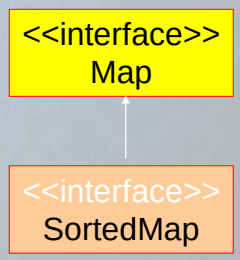
  // lấy phần tử cuối cùng (lớn nhất)
  - SortedSet subSet(Object e1, Object e2);

  // lấy một tập các phần tử nằm trong khoảng từ e1 tới e2.



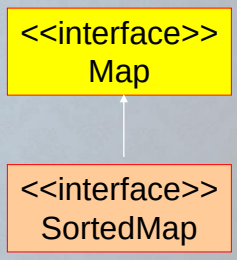
2.1. (TT). GIAO TIẾP MAP

- Giao tiếp Map cung cấp các thao tác xử lý trên các bảng ánh xạ (Bảng ánh xạ lưu các phần tử theo khoá và không có 2 khoá trùng nhau).
- 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);
  - ...



2.1. (TT). GIAO TIẾP MAP

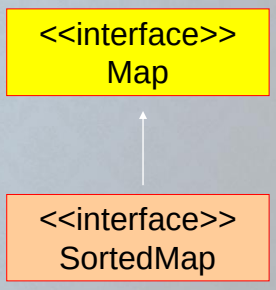
- Map cung cấp 3 cách xem dữ liệu:
  - xemcác khoá:
    Set keySet(); // Trả về các khoá
  - Xem các giá trị:
    Collection values(); // Trả về các giá trị
  - Xem các cặp khoá-giá trị
    Set entrySet(); // Trả về các cặp khoá-giá trị



- Sau khi nhận được kết quả là một collection, ta có thể dùng iterator để duyệt các phần tử của tập hợp.

2.1. (TT). GIAO TIẾP SORTEDMAP

- Giao tiếp SortedMap kế thừa từ Map.SortedMap cung cấp thao tác trên bảng ánh xạ theo 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.



2.1. (TT). HASHCODE()

- Phương thức hashCode() là một phương thức đã được định nghĩa trong lớp Object để tính các giá trị băm (hash). Vì vậy tất cả các đối tượng đều có mã mặc định của phương thức này.
- Một giá trị băm (hash) có thể là kết quả của hàm băm từ 2,3 hoặc nhiều hơn giá trị để phân biệt các đối tượng
- Nên định nghĩa phương thức hashCode() cho các đối tượng khi thao tác với tập hợp. Phương thức này nên trả về giá trị là một số nguyên (cũng có thể là số âm).

```
1  public class Item{
2      ///. . .
3      public int hashCode(){
4      return 13 * description.hashCode() + 17 * partNumber;
5      }
6      //. . .
7      private String description;
8      private int partNumber;
9  }
```

21

2.1. (TT). EQUALS()

- Hơn nữa, để so sánh hai đối tượng ta nên định nghĩa thêm hàm equals().
- Lớp Object đã định nghĩa so sánh bằng, tuy nhiên chỉ đúng đối với các lớp nguyên thủy. Còn với lớp đối tượng ta nên tự định nghĩa quy định 2 đối tượng bằng nhau khi nào

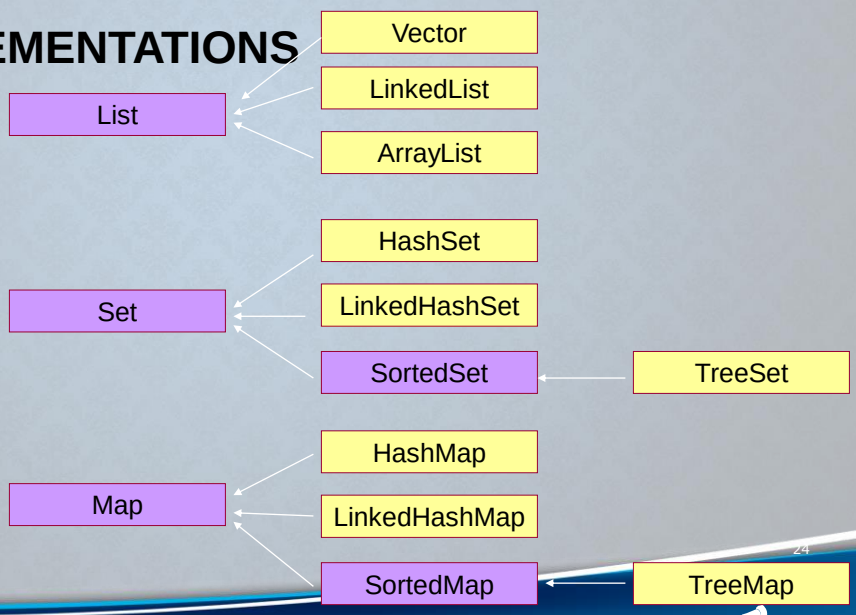
```
1  public class Item{
2      //. . .
3      public boolean equals(Object other){
4          if (other != null && getClass() == other.getClass()){
5              Item otherItem = (Item)other;
6              return description.equals(otherItem.description)
7                          && partNumber == otherItem.partNumber;
8          }
```

22

2.2. 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, bảng băm...

2.2. (TT). IMPLEMENTATIONS



2.2. (TT). MÔ TẢ CÁC CÀI ĐẶT SƠ BỘ.

- **ArrayList**: Mảng động, nếu các phần tử thêm vào vượt quá kích cỡ mảng, mảng sẽ tự động tăng kích cỡ (50% so với kích cỡ ban đầu- CAPACITY).
- **LinkedList**: Danh sách liên kết 2 chiều. Hỗ trợ thao tác trên đầu và cuối danh sách (FIFO)/LIFO.
- **Vector**: giống ArrayList nhưng có các phương thức đồng bộ dùng trong đa luồng. Mảng sẽ tự động tang kích thướng 100% so với kích cỡ ban đầu
- **HashSet**: Bảng băm.
- **LinkedHashSet**: Bảng băm kết hợp với LinkedList nhằm đảm bảo thứ tự các phần tử.
- **TreeSet**: Cây giá trị có sắp các phần tử theo khóa.
- **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 Map).
- **TreeMap**: cài đặt của Map theo thứ tự của khóa.

2.2. (TT). MÔ TẢ CÁC CÀI ĐẶT CHI TIẾT

- Nhóm Collection- List (giáo trình trang 182-186)
- ArrayList.
  - Được sử dụng như một mảng động để lưu trữ các phần tử.
  - Duy trì thứ tự của các phần tử được thêm vào
  - Không có tính năng đồng bộ (chỉ hoạt động trong 1 luồng)
  - Cho phép truy cập ngẫu nhiên lấy phần tử với độ nhanh do lưu dữ liệu theo chỉ mục
- Vector.
  - Tương tự như ArrayList, cũng chứa các phần tử trùng lặp
  - Có duy trì thứ tự các phần tử được thêm vào
  - Có đồng bộ các phương thức. Hoạt động tốt trong đa luồng
- LinkedList là liên kết hai chiều.
  - Mỗi phần tử trong danh sách giữ một tham chiếu đến phần tử trước nó và tham chiếu đến phần tử ngay sau nó

2.2. (TT). MÔ TẢ CÁC CÀI ĐẶT

- Nhóm Set (treeSet và HashSet)
  - HashSet:
    - Không chứa các phần tử trùng lặp
    - Lưu trữ các phần tử sử dụng bảng băm (hash table).
    - Không đảm bảo thứ tự được thêm vào
    - Cho phép chứa phần tử null
  - LinkedHashSet:
    - Tương tự như HashSet nhưng có cài đặt linkedList để duy trì thứ tự các phần tử
  - TreeSet lưu trữ các phần tử hình cây.
    - Lưu trữ các phần tử không trùng nhau
    - Duy trì theo thứ tự các phần tử tăng dần theo bộ so sánh được cung cấp
    - Không cho phép chứa phần tử null
    - Không đồng bộ. Không cho phép phần tử null
    - TreeSet sử dụng TreeMap lưu trữ các phần tử.

2.2. (TT). MÔ TẢ CÁC CÀI ĐẶT

- Nhóm MAP: lưu trữ dữ liệu dưới dạng cặp key và value
  - HashMap.
    - HashMap chỉ chứa các key duy nhất.
    - HashMap có thể có 1 key là null và nhiều giá trị null.
    - HashMap duy trì các phần tử **KHÔNG** theo thứ tự chèn.
  - TreeMap.
    - TreeMap chỉ chứa các key duy nhất.
    - TreeMap KHÔNG cho phép key là null và nhưng có thể có nhiều value (giá trị) null.
    - TreeMap duy trì các phần tử được thêm vào theo thứ tự key tăng dần.
  - HashTable.
    - Hashtable chứa các key duy nhất.
    - Hashtable KHÔNG cho key hoặc giá trị là null.
    - Hashtable được đồng bộ (synchronized).

2.3. LỚP TIỆN ÍCH COLLECTIONS.

- Collections là lớp tiện ích của hệ thống:
  - Cung cấp nhiều phương thức static chính là các thuật toán đa hình thao tác trên các đối tượng của Collection.
  - Các phương thức của lớp trả về ngoại lệ NullPointerException nếu các Collection hoặc các đối tượng lớp cung cấp cho chúng là null.

2.3. LỚP TIỆN ÍCH COLLECTIONS.

- Collections là lớp tiện ích của hệ thống:
  - Các phép toán trong lớp tiện ích hay dung (tt)
    - Collections.sort(…): Sắp xếp trong tập hợp
    - Collections.max(…) : Tìm kiếm phần tử lớn nhất
    - Collections.min(…) : Tìm kiếm phần tử nhỏ nhất
    - Collections.addAll(…) gộp bộ sưu tập theo chỉ định.
    - static int binarySearch(List list, Object key) – tìm kiếm nhị phân
    - static void shuffle(List list) – trộn
    - các phương thức tạo synchronized collection – đồng bộ
    - các phương thức tạo read-only collection

2.3. LỚP TIỆN ÍCH COLLECTIONS.

- **Sắp xếp trong tập hợp Collections.sort. (trang 199 giáo trình)**
  - Được sử dụng để sắp xếp các phần tử của bộ sưu tập đã cho.
  - Thứ tự sắp mặc định tăng dần hoặc theo thứ tự được sắp xếp bởi Comparator chỉ định hoặc cài đặt interface I**nterable**.
    - public static void sort(List list);// yêu cầu phần tử List phải thực thi interface **Comparable** và ghi rõ trong compareTo.
    - public static void sort(List list, **Comparator** c); //so sánh thông qua đối tượng của interface Comparator.

2.3. LỚP TIỆN ÍCH COLLECTIONS.

- Ví dụ:

•Sắp xếp tập hợp phần tử nguyên thủy

```
ArrayList<Integer> songuyen = new ArrayList();
sử dụng Collections.sort(songuyen);
```

•Sắp theo bộ chỉ định Comparator SỬ DỤNG Collections.sort(List list, **Comparator** c);

```
Comparator<TênLớp> c=new  Comparator<TênLớp >() {
        public int compare (TênLớp o1, TênLớp o2) {
                //các xử lý trả về giá trị nguyên: >0 hoặc <0 hoặc =0
                return int;
        } };
```

2.3. LỚP TIỆN ÍCH COLLECTIONS.

- Tìm kiếm Tìm kiếm nhị phân- binarySearch
  - public static int binarySearch(List list, T key);
  - Tìm kiếm một phần tử bằng phương pháp tìm kiếm nhị phân. Vậy tập hợp phải được sắp xếp trước đó
- Ví dụ:

```
ArrayList songuyen = new ArrayList();
songuyen.add(1); songuyen.add(7);songuyen.add(3); songuyen.add(2);
Collections.sort(songuyen);
int key = 3;
int vitri = Collections.binarySearch(songuyen, key);
System.out.println("Tìm thấy "+ key+ " tại vị trí thứ " + vitri);
```


2.3. LỚP TIỆN ÍCH COLLECTIONS.

- Phương thức trộn Shuffle
  - public static void shuffle(List list);
  - Xáo trộn các phần tử trong danh sách một cách ngẫu nhiên
  - Ví dụ:

```
ArrayList songuyen = new ArrayList();
songuyen.add(1); songuyen.add(7);songuyen.add(3); songuyen.add(2);
Collections.shuffle(songuyen);
System.out.println(songuyen);
```

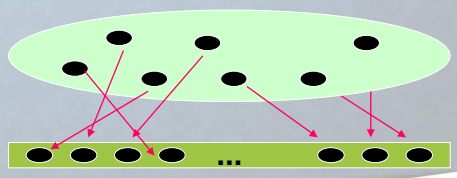
2.3. LỚP TIỆN ÍCH COLLECTIONS.

- Tìm kiếm phần tử trong tập hợp Collections.min/max
  - Sử dụng phương thức min, max trong Collections:
    - public static T min(ArrayList list);
    - public static T min(ArrayList list, Comparator c);
  - Trả về phần tử lớn (nhỏ) nhất ở trong danh sách mảng có kiểu là T.
  - Tuy nhiên muốn sắp xếp danh sách mảng đối tượng tự định nghĩa, thì phải cài đặt bộ so sánh

3. DUYỆT TẬP HỢP VỚI ITERATOR.

- Duyệt các phần tử (tr 188 giáo trình)
  - Interface Iterator cung cấp một số các phương thức để duyệt (lặp) qua các phần tử bất kỳ tập hợp nào.
  - Mỗi interface Collection đều chứa một phương thức **iterator** để trả về một thực thể của interface Iterator

Collection c;



Iterator it = c.iterator();

3. DUYỆT TẬP HỢP VỚI ITERATOR.

- Duyệt các phần tử trong tập hợp sử dụng Interator (giáo trình trang 188)
  - Giao diện Interable <Element>
    - Interator<Element> interator(): trả lại 1 đối tượng duyệt tuần tự các phần tử trong tập hợp
  - Giao diện Interator<Element>
    - Phương thức duyệt
      - **Boolean hasNext():** Trả về true nếu còn phần tử chưa được duyệt
      - **Object next()** trả về phần tử tiếp theo trong tập hợp
      - Void remove();
  - Ví dụ

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


3. DUYỆT TẬP HỢP VỚI ITERATOR.

```
ArrayList<String> myList = new  ArrayList<String>();
myList.add("a");
myList.add("b");
myList.add("c");
```

Duyệt tập hợp theo kiểu thông thường

```
for (int i = 0; i < myList.size(); i++) {
        System.out.println(myList.get(i));
}
```

```
Iterator<String> mt = myList.iterator();
while(mt.hasNext())    {
    System.out.println(mt.next());
  }
```

4. MINH HỌA

- Interface List
  - Minh họa các thao tác trong ArrayList.
  - Các lớp cài đặt khác thực hiện tương tự
- Interface Set
  - Minh họa thực hiện tương tự List
- Interface Map
  - Minh họa thực hiện tương tự như List và Set

4. MINH HỌA

- ArrayList
  - Lớp ArrayList nằm trong gói java.util, triển khai interface List.
  - Chức năng tương tự như mảng, duy trì phần tử theo thứ tự nhập vào;
  - Cho phép các phần tử trùng nhau, không chứa phần tử trống.
  - Kích thước ArrayList có thể lớn lên và nhỏ đi trong thời gian sống của nó
  - Có 3 cách tạo ArrayList
    - new ArrayList(); - tạo tổng quát
    - new ArrayList<T>(int n); - tạo tường minh với sức chứa ban đầu
    - new ArrayList<T>(Collection c); - tạo thông qua các phần tử tập hợp khác

4. MINH HỌA

- ArrayList
  - Các phép toán triển khai interface List:
    - public boolean **add(Object** element);
    - public boolean **add(int** index, Object element);
    - public int **size**(); Trả về số lượng phần tử có trong danh sách .
    - public Object **get**(int index); lấy phần tử tại vị trí index
    - public int **indexOf(Object** o);Trả lại vị trí của phần tử o;
    - public boolean **isEmpty**(); Kiểm tra danh sách rỗng
    - public void **set(int** index, Object o); Sửa một phần tử ở vị trí thứ index

4. MINH HỌA

- ArrayList
  - Các phép toán hữu ích khác
    - public Object[] **toArray**(); Chuyển đổi danh sách mảng sang mảng thường:
    - public void **clear**(); Xóa hết các các phần tử trong ds
    - public int **remove(Object** e); Xóa phần tử gần nhất có giá trị bằng e và trả về vị trí của phần tử đó:
    - public Object **contains(Object** e); Kiểm tra object o có trong tập hợp không. Để phương thức thực hiện đúng cần ghi đè phương thức **equal** để xác định tiêu chí so sánh.

4. MINH HỌA- BÀI TẬP ÁP DỤNG 1

Yêu cầu đầu bài

- Cho 1 dãy số nguyên. Sử dụng các thao tác trong tập hợp thực hiện:
  - nhập thêm phần thử không trùng nhau
  - xóa phần tử
  - tìm phần tử nhỏ nhất
  - sắp xếp phần tử
  - in dãy số

Các phép toán được dùng

- add(phần thử)
- remove(vị trí)
- contain(phần tử): kiểm tra tồn tại trước khi thực hiện
- Collections.min()
- Collections.max()
- Interator
- size()

4. MINH HỌA- BÀI TẬP ÁP DỤNG 1

Cấu trúc hàm main

```
4   import java.util.Collections;
5   import java.util.Iterator;
6   import java.util.Scanner;
7   public class Arr_Mang {
        static ArrayList<Integer> dayso = new ArrayList<Integer>();
9       public static void nhapDS(){...13 lines }
22      public static void inDaySo(){...6 lines }
28      public static void sapDaySo(){...5 lines }
33      public static int timMin()  {...5 lines }
38      public void xoaPhanTu(int vitri){...10 lines }
48
49      public static void main(String []args){
50          nhapDS();
51          inDaySo();
52          System.out.println("phan tu nho nhat trong day: "+timMin());
53          sapDaySo();
54      }
55  }
```

4. MINH HỌA- BÀI TẬP ÁP DỤNG 1

Nội dung xử lý nhập dãy số với các phần tử không trùng nhau

```
4  import java.util.Collections;
5  import java.util.Iterator;
6  import java.util.Scanner;
7  public class Arr_Mang {
       static ArrayList<Integer> dayso = new ArrayList<Integer>();
9      public static void nhapDS(){
10         Scanner s = new Scanner(System.in);
11         int i, dem =0;
12         while(dem<10){
13             System.out.print("i=");
14             i=s.nextInt();
15             if (dayso.contains(i))
16                 System.out.print("phần tử đã có");
17             else
18                 dayso.add(i);
20                 dem++;
21         }
22     public static void inDaySo(){...6 lines }
28     public static void sapDaySo(){...5 lines }
33     public static int timMin()  {...5 lines }
```


4. MINH HỌA- BÀI TẬP ÁP DỤNG 1

Nội dung xử lý in dãy số

```
4  import java.util.Collections;
5  import java.util.Iterator;
6  import java.util.Scanner;
7  public class Arr_Mang {
       static ArrayList<Integer> dayso = new ArrayList<Integer>();
9      public static void nhapDS(){...13 lines }
22     public static void inDaySo(){
23         Iterator<Integer> myIterator = dayso.iterator();
24         while(myIterator.hasNext()){
25             System.out.println(myIterator.next()+", ");
26         }
27     }
28     public static void sapDaySo(){...5 lines }
33     public static int timMin()  {...5 lines }
38     public void xoaPhanTu(int vitri){...10 lines }
48
49     public static void main(String []args){
50         nhapDS();
51         inDaySo();
52         System.out.println("phan tu nho nhat trong day: "+timMin());
53         sapDaySo();
```

4. MINH HỌA- BÀI TẬP ÁP DỤNG 1

Nội dung xử lý sắp xếp dãy số

```
4  import java.util.Collections;
5  import java.util.Iterator;
6  import java.util.Scanner;
7  public class Arr_Mang {
       static ArrayList<Integer> dayso = new ArrayList<Integer>();
9      public static void nhapDS(){...13 lines }
22     public static void inDaySo(){...6 lines }
28     public static void sapDaySo(){
29         Collections.sort(dayso);
30         System.out.println("ds sau khi sắp là");
31         inDaySo();
32     }
33     public static int timMin()  {...5 lines }
38     public void xoaPhanTu(int vitri){...10 lines }
48
49     public static void main(String []args){
50         nhapDS();
51         inDaySo();
52         System.out.println("phan tu nho nhat trong day: "+timMin());
53         sapDaySo();
54     }
```


4. MINH HỌA- BÀI TẬP ÁP DỤNG 1

Nội dung xử lý tìm phần tử nhỏ nhất

```
4  import java.util.Collections;
5  import java.util.Iterator;
6  import java.util.Scanner;
7  public class Arr_Mang {
       static ArrayList<Integer> dayso = new ArrayList<Integer>();
9      public static void nhapDS(){...13 lines }
22     public static void inDaySo(){...6 lines }
28     public static void sapDaySo(){...5 lines }
33     public static int timMin()  {
34         int min = Collections.min(dayso);
35         System.out.println("Phan tu: "+min);
36         return min;
37     }
38     public void xoaPhanTu(int vitri){...10 lines }
48
49     public static void main(String []args){
50         nhapDS();
51         inDaySo();
52         System.out.println("phan tu nho nhat trong day: "+timMin());
53         sapDaySo();
54     }
```

4. MINH HỌA- BÀI TẬP ÁP DỤNG 1

Nội dung xử lý xóa phần tử tại vị trí chỉ định có kiểm tra vị trí không ngoài chỉ số

```
4   import java.util.Collections;
5   import java.util.Iterator;
6   import java.util.Scanner;
7   public class Arr_Mang {
        static ArrayList<Integer> dayso = new ArrayList<Integer>();
9       public static void nhapDS(){...13 lines }
22      public static void inDaySo(){...6 lines }
28      public static void sapDaySo(){...5 lines }
33      public static int timMin()  {...5 lines }
38      public void xoaPhanTu(int vitri){
39          System.out.println("Nhap vi tri: ");
40          if((vitri<0)||(vitri>dayso.size())){
41              System.out.println("Chi so khong dung");
42          }else{
43              dayso.remove(vitri);
44              System.out.println("Danh sach sau xoa:");
45              System.out.println(dayso);
46          }
47      }
```


4. (TT) MINH HỌA ARRAYLIST VỚI ĐỐI TƯỢNG TỰ TẠO.

Minh họa Arraylist với đối tượng tự tạo.

```
9   public class Nguoi {
        private String soCMND;
11      private String ten;
12      private int tuoi;
13
14      public Nguoi(String soCMND, String ten, int tuoi) {...5 lines }
19      public Nguoi(String soCMND) {...3 lines }
22      public String getTen() {...3 lines }
25      public int getTuoi() {...3 lines }
28      @Override
        public String toString() {...3 lines }
32      @Override
        public boolean equals(Object obj) {
34          final Nguoi other = (Nguoi) obj;
            if (!Objects.equals(this.soCMND, other.soCMND)) {
36              return false;
37          }
38          return true;
39      }
```

4. (TT)MINH HỌA ARRAYLIST VỚI ĐỐI TƯỢNG TỰ TẠO

Hàm tạo đối tượng

```
9   public class Nguoi {
        private String soCMND;
11      private String ten;
12      private int tuoi;
13
14      public Nguoi(String soCMND, String ten, int tuoi) {
15          this.soCMND = soCMND;
16          this.ten = ten;
17          this.tuoi = tuoi;
18      }
19      public Nguoi(String soCMND) {
20          this.soCMND = soCMND;
21      }
22      public String getTen() {...3 lines }
25      public int getTuoi() {...3 lines }
```


4. (TT) MINH HỌA ARRAYLIST VỚI ĐỐI TƯỢNG TỰ TẠO

ghi đè hàm equal

```
9   public class Nguoi {
        private String soCMND;
11      private String ten;
12      private int tuoi;
13
14      public Nguoi(String soCMND, String ten, int tuoi) {...5 lines }
19      public Nguoi(String soCMND) {...3 lines }
22      public String getTen() {...3 lines }
25      public int getTuoi() {...3 lines }
28      @Override
        public String toString() {...3 lines }
32      @Override
        public boolean equals(Object obj) {
34          final Nguoi other = (Nguoi) obj;
            if (!Objects.equals(this.soCMND, other.soCMND)) {
36              return false;
37          }
38          return true;
39      }
```


4. (TT) MINH HỌA ARRAYLIST VỚI ĐỐI TƯỢNG TỰ TẠO

các nội chính trong main

```
12  public class NguoiDemo {
    static ArrayList<Nguoi> list=new ArrayList<Nguoi>();
14  static void nhapDS(){...7 lines }
21  static void inDS(){...7 lines }
28  static void sapXep(){...13 lines }
41  static void timKiem(String cmtndTim){...10 lines }
51  public static void main(String[] args) {
52      Scanner s=new Scanner(System.in);
53      nhapDS();
54      System.out.println("ds sau khi nhap");
55      inDS();
56      String cmtndTim=s.nextLine();
57      timKiem(cmtndTim);
58      sapXep();
59      System.out.println("DS sau khi săp:");
60      inDS();
61     }
```


4. (TT) MINH HỌA ARRAYLIST VỚI ĐỐI TƯỢNG TỰ TẠO –

triển khai nhập danh sách

```
12  public class NguoiDemo {
    static ArrayList<Nguoi> list=new ArrayList<Nguoi>();
14  static void nhapDS(){
15      Nguoi p1=new Nguoi("001","Hong",13);
16      Nguoi p2=new Nguoi("002","lan2",13);
17      Nguoi p3=new Nguoi("003","Anh",13);
18      Nguoi p4=new Nguoi("004","lan4",34);
19      list.add(p4); list.add(p2);list.add(p3);list.add(p1);
20  }
21  static void inDS(){...7 lines }
28  static void sapXep(){...13 lines }
41  static void timKiem(String cmtndTim){...10 lines }
51  public static void main(String[] args) {
52      Scanner s=new Scanner(System.in);
53      nhapDS();
54      System.out.println("ds sau khi nhap");
55      inDS();
56      String cmtndTim=s.nextLine();
```

4. (TT) MINH HỌA ARRAYLIST VỚI ĐỐI TƯỢNG TỰ TẠO –

In tập hợp sử dụng Iterator

```
12  public class NguoiDemo {
    static ArrayList<Nguoi> list=new ArrayList<Nguoi>();
14  static void nhapDS(){...7 lines }
21  static void inDS(){
22      Iterator<Nguoi> myIterator =list.iterator();
23      while(myIterator.hasNext())
24      {
25          System.out.print(myIterator.next()+",");
26      }
27  }
28  static void sapXep(){...13 lines }
41  static void timKiem(String cmtndTim){...10 lines }
51  public static void main(String[] args) {
52      Scanner s=new Scanner(System.in);
53      nhapDS();
54      System.out.println("ds sau khi nhap");
55      inDS();
56      String cmtndTim=s.nextLine();
57      timKiem(cmtndTim);
```


4. (TT) MINH HỌA ARRAYLIST VỚI ĐỐI TƯỢNG TỰ TẠO –

sử cụng Comparator để tạo bô so sánh

```
12  public class NguoiDemo {
    static ArrayList<Nguoi> list=new ArrayList<Nguoi>();
14  static void nhapDS(){...7 lines }
21  static void inDS(){...7 lines }
28  static void sapXep(){
29      Comparator<Nguoi> c=new Comparator<Nguoi>() {
30          @Override
            public int compare(Nguoi t, Nguoi t1) {...7 lines }
38      };
39       Collections.sort(list, c);
40  }
41  static void timKiem(String cmtndTim){...10 lines }
51  public static void main(String[] args) {
52      Scanner s=new Scanner(System.in);
53      nhapDS();
54      System.out.println("ds sau khi nhap");
55      inDS();
56      String cmtndTim=s.nextLine();
57      timKiem(cmtndTim);
```

4. (TT) MINH HỌA ARRAYLIST VỚI ĐỐI TƯỢNG TỰ TẠO –

sử dụng contains và indexOf

```
14    static void nhapDS(){...7 lines }
21    static void inDS(){...7 lines }
28    static void sapXep(){...13 lines }
41    static void timKiem(String cmtndTim){
42        System.out.print("nhap scmtnd can tim");
43        Nguoi pTim=new Nguoi(cmtndTim);
44        if (list.contains(pTim))
45                System.out.print(pTim+" thay o vi tri"+
46                        list.indexOf(pTim));
47        else
48                System.out.print("khong tim duoc thong tin");
49
50    }
51    public static void main(String[] args) {
52        Scanner s=new Scanner(System.in);
53        nhapDS();
54        System.out.println("ds sau khi nhap");
55        inDS();
56        String cmtndTim=s.nextLine();
57        timKiem(cmtndTim);
```


4. MINH HỌA- BÀI TẬP ÁP DỤNG 2

- Thông tin Thí sinh dự thi được mô tả thông qua: SBD, tên thí sinh, điểm toán, điểm lý, điểm hóa. Yêu cầu tên, mã không được nhập trống, điểm phải nằm trong khoảng 0-10. SBD không được trùng nhau
- Mỗi thí sinh dự thi được xếp vào phòng thi. Thông tin về phòng thi bao gồm: mã phòng thi, địa điểm thi, số lượng thí sinh trong phòng, danh sách thí sinh
- Cài đặt lớp chứa hàm main thực hiện 8 yêu cầu:

4. MINH HỌA- BÀI TẬP ÁP DỤNG 2

- Thực hiện tại 1 điểm thi, Hiển thị menu cho chọn:

1. Nhập danh sách phòng thi. Nhập danh sách thi sinh trong phòng. (chú ý số thí sinh không vượt quá số chỗ trong phòng, bắt lỗi dữ liệu)
2. In danh sách sau khi nhập theo dạng bảng
3. Nhập mã thí sinh. Kiểm tra ts có trong ds không. Xóa thí sinh có mã vừa nhập
4. Nhập mã thí sinh. Kiểm tra ts có trong ds không. Sửa thông tin thí sinh có mã vừa nhập
5. Tìm kiếm 1 thí sinh theo mã hay theo vị trí. Hiển thị TS tìm
6. Sắp xếp thí sinh trong phòng tăng dần theo tên. Tên trùng nhau sắp theo mã
7. Đưa ra TS có tổng điểm thi cao nhất
8. Đưa ra ts có tổng điểm thi thấp nhất

4. MINH HỌA- BÀI TẬP ÁP DỤNG 2-GỢI Ý THỰC HIỆN

Nội dung yêu cầu

- Liệt kê các nội dung trong bài toán
  - Xây dựng lớp cơ sở
  - Xây dựng lớp trung gian cài đặt quan hệ lớp
  - Sắp xếp bố trí các nội dung thực hiện theo yêu cầu bài toán

Các phép toán trong tập hợp

- Liệt kê các xử lý
  - Xử lý trong lớp
  - Xử lý của tập hợp
    - Lựa chọn tập hợp
    - Danh sách các phép toán trong tập hợp
    - Các xử lý chuyển đổi nếu có

NỘI DUNG SAU BUỔI HỌC

- Thực hiện các minh họa các lớp triển khai collection theo nhóm (tham khảo video minh họa lớp Arraylist kế thừa interface List). Làm phiếu bài tập 1.
- Chi tiết yêu cầu: Làm slide giới thiệu nội dung trình bày và cài đặt bài toán minh họa.
- Kết quả
  - Trình bày - thảo luận trong buổi học trên lớp.