Bài 7
Generic và Collection
File

MỤC TIÊU

- Giới thiệu Generics
- Giới thiệu Collections
- List và ArrayList
- Set và SortedSet
- Map và HashMap
- Đọc và ghi file trong java

Giới thiệu Generics

- **Generic là gì?**
  - Là tính năng của Java cho phép người lập trình có thể chỉ định rõ kiểu dữ liệu mà họ muốn làm việc với một method, class và interface
- **Tại sao cần Generic?**
  - Xây dựng phương thức, lớp, interface có tính chất tương tự nhau nhưng trên các kiểu dữ liệu khác nhau
- **Quy ước đặt tên kiểu tham số Generic**
  - E – Element
  - K – Key
  - V – Value
  - N – Number
  - T – Type – Kiểu dữ liệu bất kỳ thuộc Wrapper Class: String, Integer, Long, Float...
  - S,U,V.... – Các kiểu dữ liệu khác
- **Ký tự Diamond <>**
  - Từ Java 7 trở lại đây, có thể thay thế các đối số kiểu dữ liệu cần thiết để gọi hàm khởi tạo của một lớp Generic bằng ký tự Diamond

```java
package bkap_generic_demo;

/**
 *
 * @author QUANGND
 */
public interface GenericInterfaceDemo<T> {

    public void insert(T obj);

    public void update(T obj);
}
```

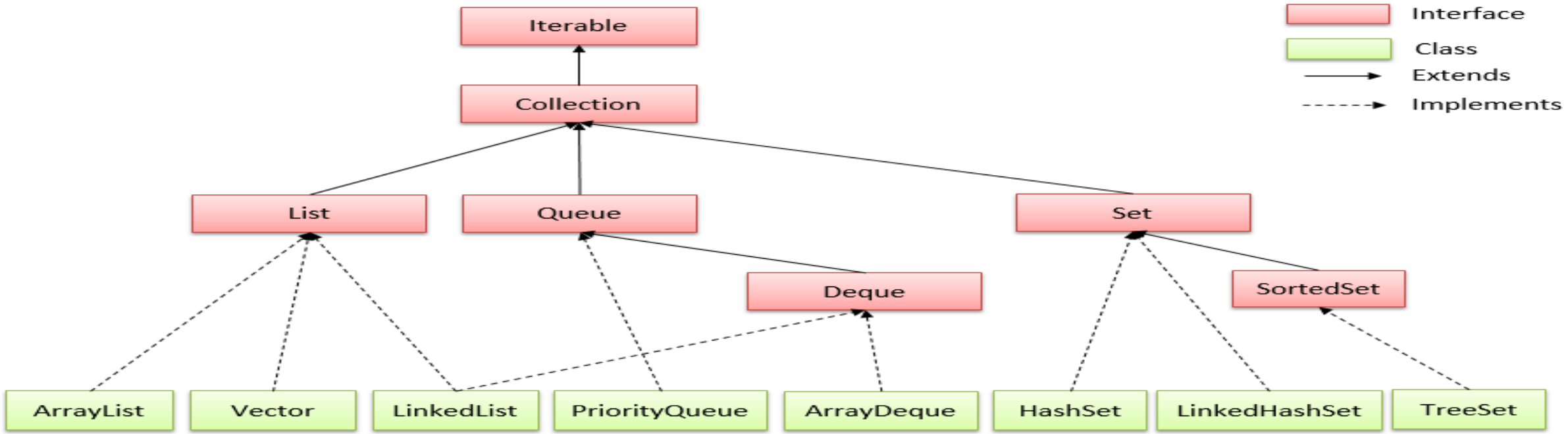
```java
package bkap_generic_demo;

/**
 *
 * @author QUANGND
 */
public class GenericMethodDemo {

    public static <T> void printArray(T[] arrayT) {
        for (T t : arrayT) {
            System.out.printf("%s\t", t);
        }
        System.out.printf("\n");
    }
}
```

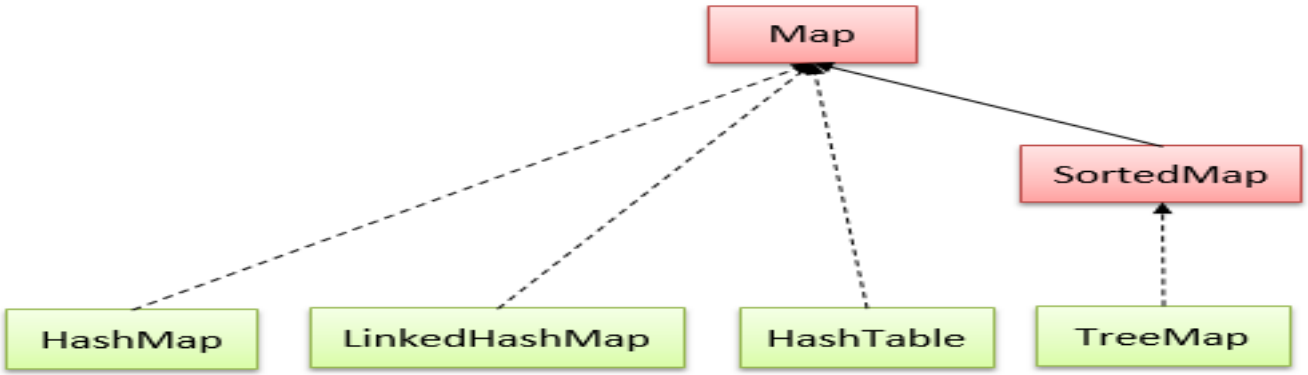
```java
package bkap_generic_demo;

/**
 *
 * @author QUANGND
 */
public class GenericClassDemo<K,V> {
    private K key;
    private V value;

    public K getKey() {
        return key;
    }

    public void setKey(K key) {
        this.key = key;
    }

    public V getValue() {
        return value;
    }

    public void setValue(V value) {
        this.value = value;
    }

}
```

Giới thiệu Collections

- **Các hạn chế khi sử dụng mảng**
  - Mảng có kích cỡ và số chiều cố định nên rất khó khan cho việc mở rộng mảng
  - Các phần tử được đặt và tham chiếu một cách liên tiếp nhau trong bộ nhớ nên khó khan cho việc xóa một phần tử ra khỏi mảng (Mất tính liên tiếp, giảm hiệu năng của chương trình)
- **Collections – Tập hợp**
  - Tập hợp các lớp dùng để lưu trữ danh sách và có khả năng tự co dãn khi danh sách đó thay đổi
- **Collections Framework**
  - Framework
    - Là một tập hợp các thư viện (library) đã được đóng gói để hỗ trợ phát triển ứng dụng trên framework đó.
    - Cung cấp các nguyên tắc, cấu trúc của ứng dụng mà chúng ta phải tuân thủ theo nó
  - Collections Framework
    - Là tập hợp các lớp (Class) và các interface dùng để hỗ trợ việc thao tác trên tập các đối tượng
    - Gồm các thành phần:
      - ✓ Interface
      - ✓ Implementations: thành phần triển khai các interface (Class)
      - ✓ Algorithms: các phương thức dùng để thực thi các phép toán như tìm kiếm, sắp xếp trên các đối tượng

Giới thiệu Collections





Giới thiệu Collections

Interfaces	
Tên Interface	**Đặc điểm**
List	Các phần tử trong List interface được sắp xếp có thứ tự và có thể có giá trị giống nhau
Set	Các phần tử trong Set là duy nhất – giá trị các phần tử không được giống nhau
Sorted Set	Kế thừa từ Set, giá trị các phần tử mặc định được sắp xếp tăng dần
Map	Mỗi phần tử là một cặp <Key,Value>, Key là duy nhất
SortedMap	Kế thừa Map, Key được sắp xếp tang dần

Class	
Tên Class	**Đặc điểm**
LinkedList	Cấu trúc dữ liệu lưu trữ các phần tử dưới dạng danh sách. Các phần tử được sắp xếp có thự tự và có thể giống nhau
ArrayList	Sử dụng cấu trúc mảng để lưu trữ phần tử. Thứ tự các phần tử dựa theo thứ tự lúc thêm vào và giái trị các phần tử có thể trùng nhau
HashSet	Thứ tự các phần tử không dựa vào thứ tự lúc thêm vào và giá trị các phần tử phải là duy nhất
TreeSet	Các phần tử được mặc định sắp xếp tang dần và các phần tử phải là duy nhất
HashMap	Phần tử là một cặp <Key,Value> và Key là duy nhất. Cho phép truy xuất trực tiếp dữ liệu thông qua Key
TreeMap	Phần tử là một cặp <Key,Value> và Key là duy nhất. Giá trị các phần tử được sắp xếp tăng dần

Giới thiệu Collections

- **Thao tác các phần tử trong collections**
  - Sử dụng vòng lặp
  - Sử dụng **Iterator** và **ListIterator**
- **Các phương thức khai báo trong Iterator**
  - ❑ **boolean hastNext():** Trả về true nếu còn phần tử, ngược lại là false
  - ❑ **Object next():** Trả về phần tử kế tiếp. Nếu không có phần tử kế tiếp ném ra NoSuchElementException
  - ❑ **void remove():** Xóa phần tử hiện tại

```
6   package bkap_iterator_demo;
7
8   import java.util.ArrayList;
9   import java.util.Iterator;
10  import java.util.List;
11
12  /**
13   *
14   * @author QUANGND
15   */
16  public class BKAP_Iterator_Demo {
17
18      /**
19       * @param args the command line arguments
20       */
21      public static void main(String[] args) {
22          //Khoi tao danh sach ten sinh vien
23          List<String> listStudentName = new ArrayList<>();
24          listStudentName.add("Nguyen Van A");
25          listStudentName.add("Nguyen Van B");
26          listStudentName.add("Nguyen Van C");
27          //Su dung iterator de hien thi ten cac sinh vien
28          Iterator<String> itr = listStudentName.iterator();
29          System.out.println("Ten cac sinh vien: ");
30          while (itr.hasNext()) {
31              String studentName = itr.next();
32              System.out.println(studentName);
33          }
34      }
35  }
```

```
Output - BKAP_Iterator_Demo (run)
run:
Ten cac sinh vien:
Nguyen Van A
Nguyen Van B
Nguyen Van C
BUILD SUCCESSFUL (total time: 0 seconds)
```

List và ArrayList

- **Interface List** là một phần trong Collection.
- Nó cho phép thêm các đối tượng (**kể cả trùng lặp**) vào danh sách.
- List cho phép thêm phần tử vào **vị trí chỉ định**.
- List sử dụng **chỉ mục (index)** để xác định vị trí của phần tử (bắt đầu từ 0).
- List là tuần tự nên có thể truy cập bằng **iterator**.

Interface List có các phương thức:

- add(int index, E element)
- addAll(int index, Collection<? extends E> c)
- get(int index)
- set(int index, E element)
- remove(int index)
- subList(int start, int end)
- indexOf(Object o)
- lastIndexOf(Object o)

List và ArrayList

- Lớp **ArrayList** thực thi interface List.
- ArrayList là một mảng các phần tử với **kích thước có thể thay đổi**.
- Phần tử trong ArrayList có thể là **null**.
- ArrayList phù hợp với việc **truy cập ngẫu nhiên** vào phần tử bất kỳ

ArrayList có các constructor như sau:

- ArrayList()
- ArrayList(Collection <? extends E> c)
- ArrayList(int initialCapacity)

ArrayList có các phương thức:

- add(E obj)
- trimToSize()
- ensureCapacity(int minCap)
- clear()
- contains(Object obj)
- size()

Set và SortedSet

- **Interface Set** tạo ra một danh sách các đối tượng không có thứ tự.
- Set không chứa **dữ liệu trùng lặp**.
- **Kế thừa đầy đủ** các phương thức từ interface Collection.
- Về cơ bản **Set tương tự List** ngoại trừ phương thức thêm phần tử **add()** không chấp nhận giá trị trùng lặp.

Interface Set có các phương thức:

- containsAll(Collection<?> obj)
- addAll(Collection<? extends E> obj)
- retainAll(Collection<?> obj)
- removeAll(Collection<?> obj)

Set và SortedSet

- **Interface SortedSet** kế thừa interface Set và nó thực hiện sắp xếp thứ tự phần tử theo tăng dần.
- **Sắp xếp** có thể thực hiện tự động hoặc sử dụng **Comparator** khi tạo SortedSet.
- SortedSet được sử dụng khi muốn tạo ra danh sách các **phần tử không trùng lặp được sắp xếp**.

Map và HashMap

- **Map** là đối tượng lưu trữ dữ liệu dưới dạng mối quan hệ **KHÓA** và **GIÁ TRỊ**.
- Mỗi Khóa (key) sẽ nối với chỉ một giá trị (value) xác định.
- Khóa **không được trùng lặp** – phải là duy nhất.
- Map **không kế thừa** interface Collection.

Collection API có 3 cách tiếp cận với Map thông qua:

- HashMap
- TreeMap
- LinkedHashMap

Các phương thức:

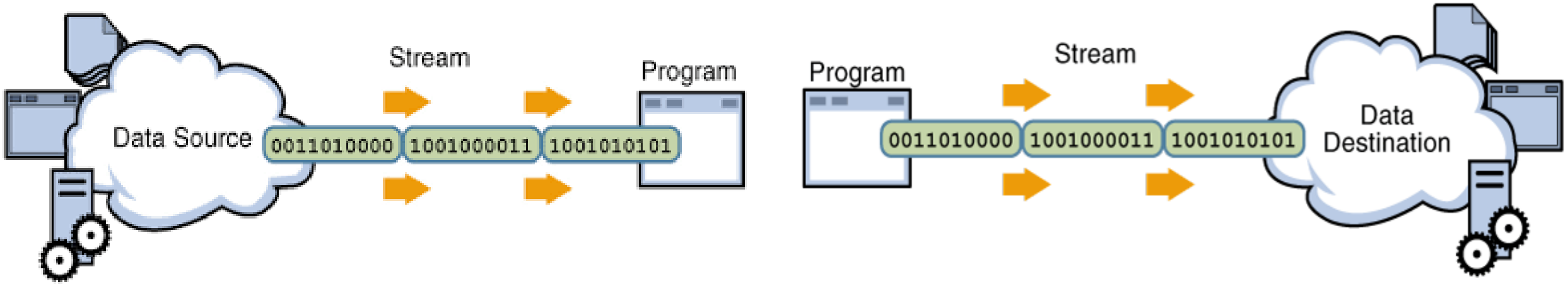
- put(K key, V value)
- get(Object key)
- containsKey(Object key)
- containsValue(Object value)
- size()
- values()

Comparable interface và Comparator interface

- **Comparable interface**
  - Sắp xếp các đối tượng của lớp do người dùng định nghĩa
  - Thuộc gói java.lang và có phương thức:
    - **int compareTo(Object obj)** – so sánh đối tượng hiện tại với đối tượng được chỉ định
- **Comparator interface**
  - Sắp xếp các đối tượng do người dùng định nghĩa
  - Thuộc gói java.util, có 2 phương thức
    - **int compare(Object obj1, Object obj2):** So sánh đối tượng obj1 với obj2
    - **boolean equals(Object obj):** so sánh bằng
- **So sánh Comparable và comparator**

Comparable	Comparator
Phải kế thừa Comparable interface	Không phải kế thừa Comparator interface
Cung cấp phương thức compareTo để sắp xếp các phần tử	Cung cấp phương thức compare để sắp xếp các phần tử
Thuộc java.lang package	Thuộc java.util package
Sử dụng Collections.sort(List) để sắp xếp	Sử dụng Collections.sort(List,Comparator) để sắp xếp

- **Data Stream**
  - **Luồng dữ liệu** hoặc thực thể logic xuất hoặc nhập thông tin
  - Kênh thông tin quan đó dữ liệu được truyền từ **nguồn** tới **đích**
  - Nguồn hoặc đích**:** **thiết bị đầu vào-ra**, **thiết bị lưu trữ**, **mạng máy tính**
- Java xây dựng nhiều loại stream khác nhau để thực hiện hoạt động **nhập/xuất**:
  - **in**: Vào tiêu chuẩn dùng để đọc dữ liệu
  - **out**: Ra tiêu chuẩn dùng để ghi dữ liệu
  - **err**: Lỗi tiêu chuẩn



- Các bước làm việc với Stream khi nhập/xuất dữ liệu

1
- Mở stream cùng với mô tả dữ liệu nguồn như: file, socket, URL…

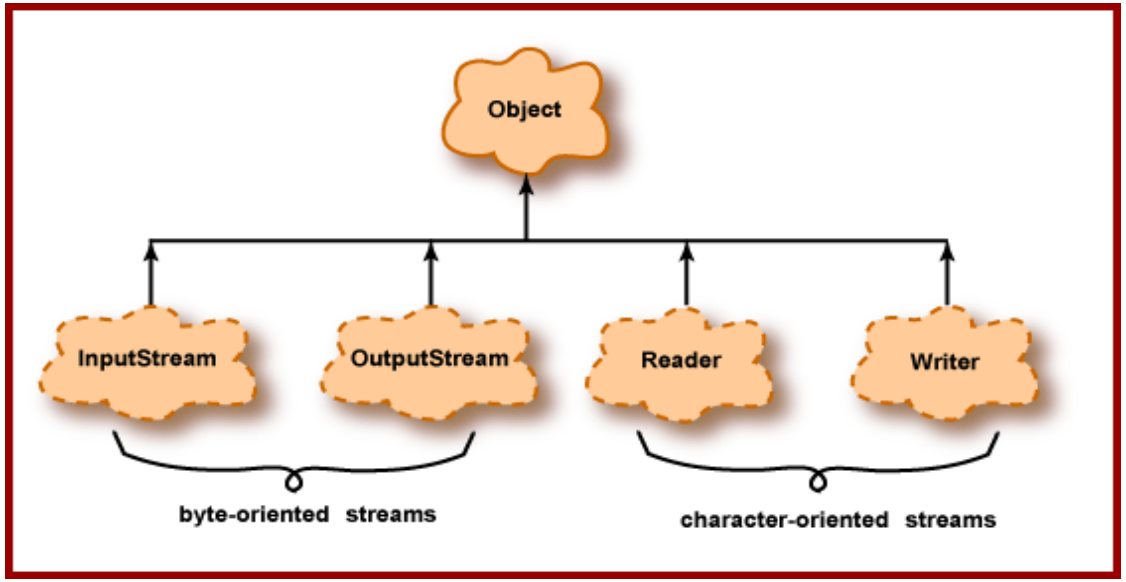
2
- Nhập/xuất dữ liệu với stream.

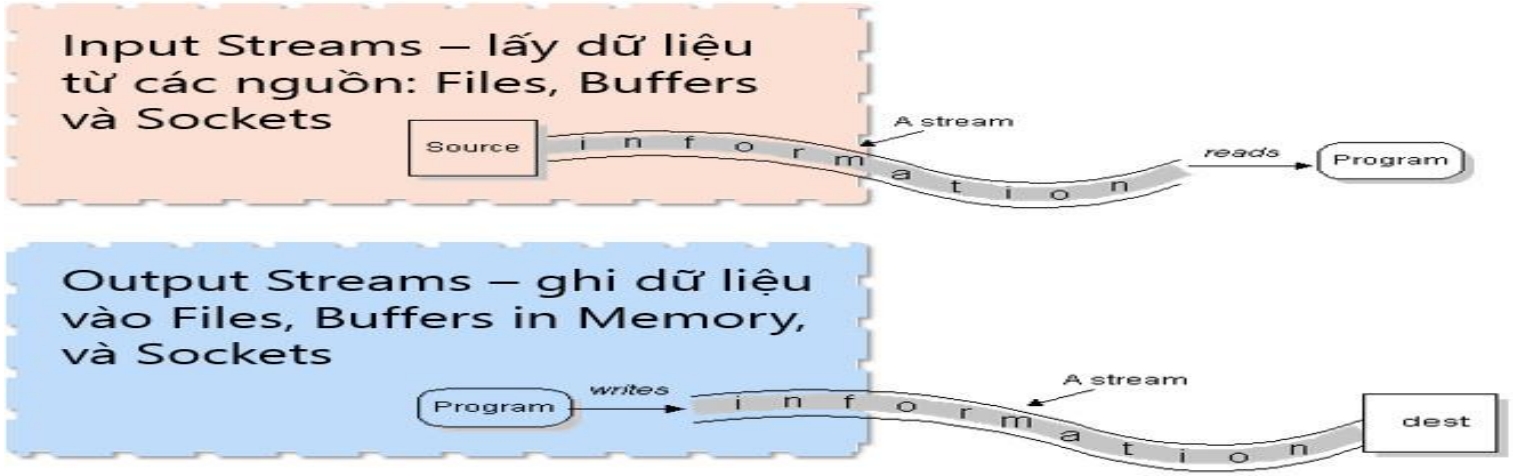
3
- Đóng stream.

File – Data Stream

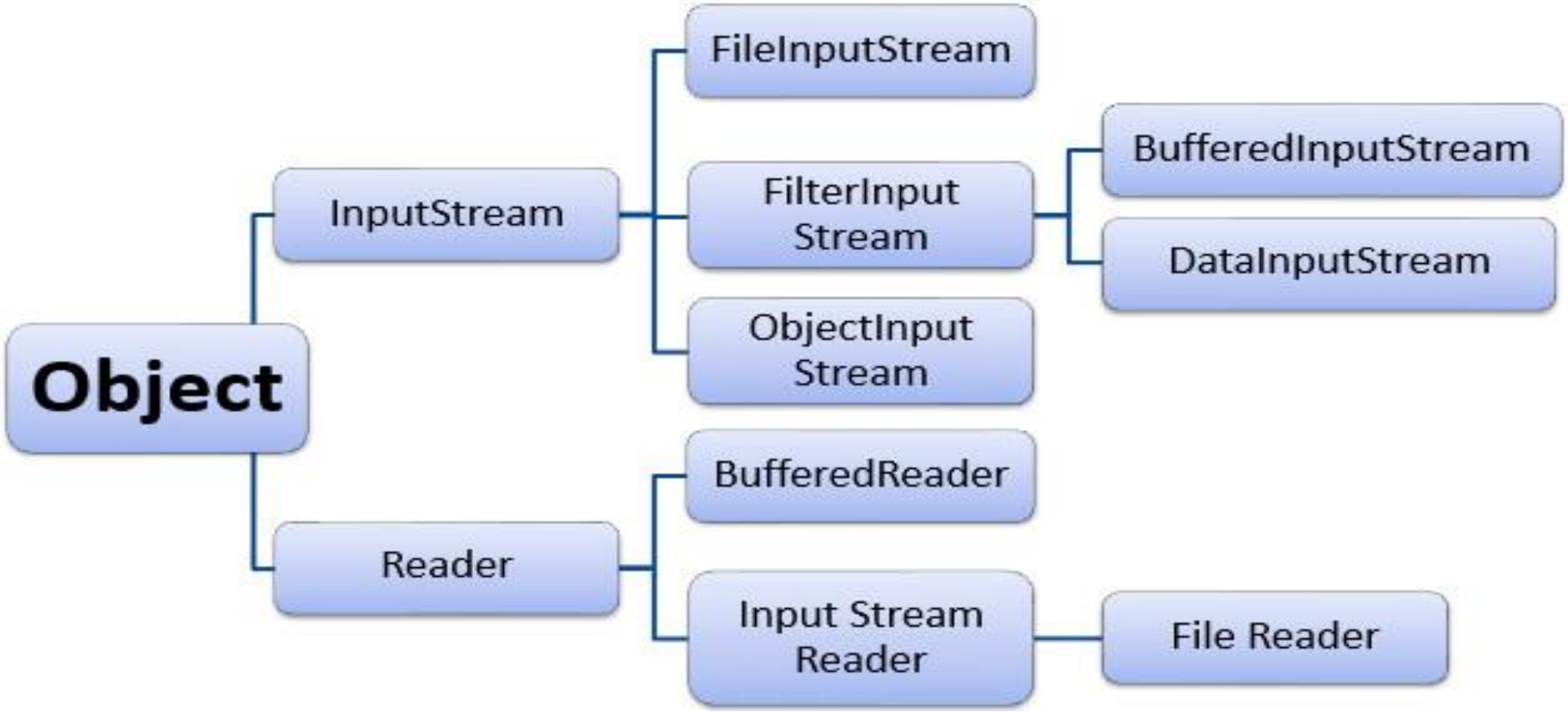
- **File**
  - Làm việc trực tiếp với file và hệ thống
  - Các phương thức lớp file cho phép tạo, xóa, đổi tên, liệt kê các thư mục
  - Các interface, lớp định nghĩa trong gói **java.nio** giúp máy ảo java truy cập hệ thống tập tin và thuộc tính
  - Các constructor của lớp File
    - File(**String** dirpath)
    - File(**String** parent, **String** child)
    - File(**File** fileobj, **String** filename)
    - File(**URL** urlobj)
  - Các phương thức lớp File
    - **renameTo(File newname)**: đổi tên file có sẵn bằng tên mới với tham số tên.
    - **delete():** xóa file
    - **exists():** kiểm tra file hoặc thư mục có tồn tai không
    - **getPath():** lấy về đường dẫn tới file/thư mục.
    - **isFile():** kiểm tra xem có phải là file không
    - **createNewFile():** tạo file mới (với đường dẫn chỉ định) nếu không có file nào tên tương tự
    - **mkdir():** tạo thư mục mới
    - **toPath():** trả về đối tượng java.nio.file.Path
    - **toURI():** xây dựng tệp, URI. Tập tin này đại diện cho tên đường dẫn trừu tượng

- **Các loại luồng dữ liệu**
  - **Luồng byte**:
    - Hỗ trợ việc nhập/xuất dữ liệu trên byte
    - Thường được dùng khi đọc ghi **dữ liệu nhị phân**
    - Gồm 2 lớp trừu tượng: InputStream và OutputStream
  - **Luồng character**:
    - Hỗ trợ nhập/xuất dữ liệu kiểu **ký tự Unicode**
    - Gồm 2 lớp trừu tượng: Reader và Writer

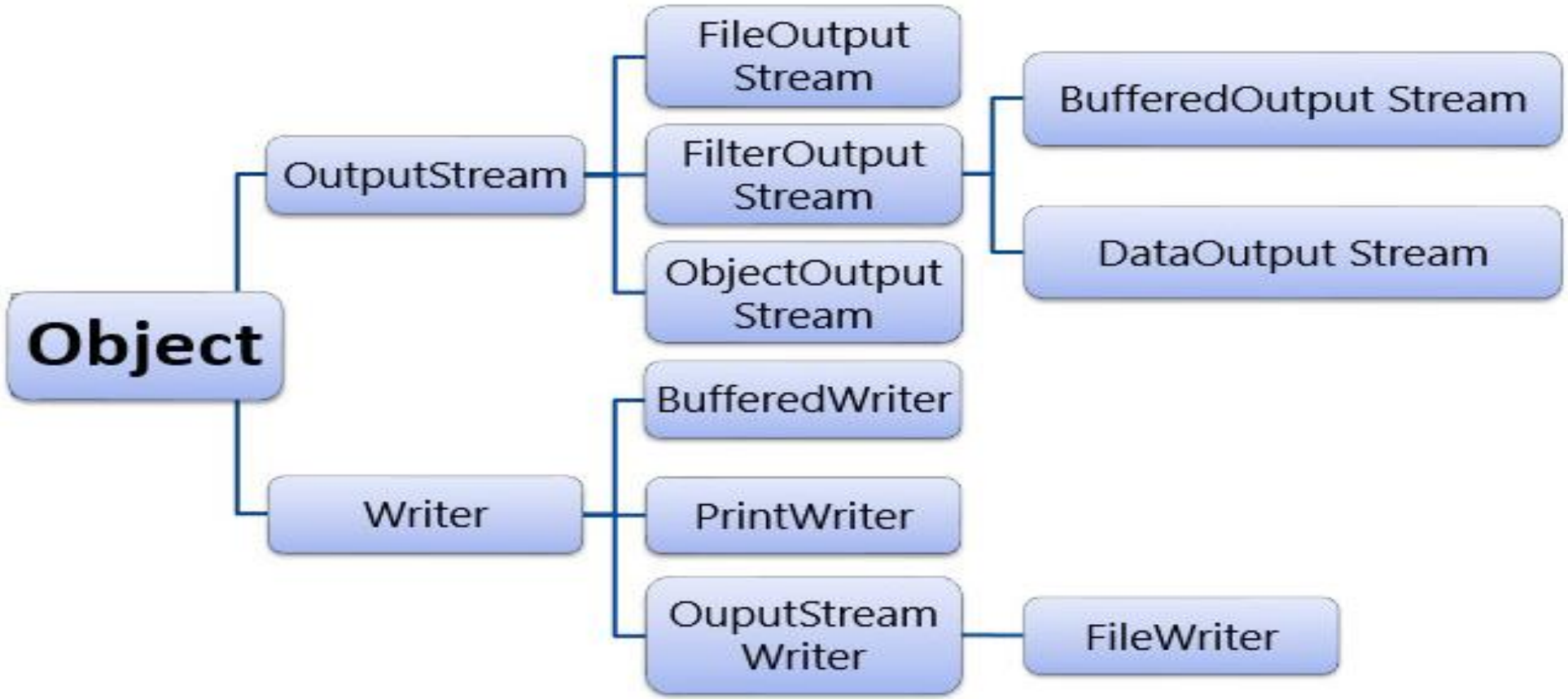




File – Data Stream – Luồng nhập dữ liệu

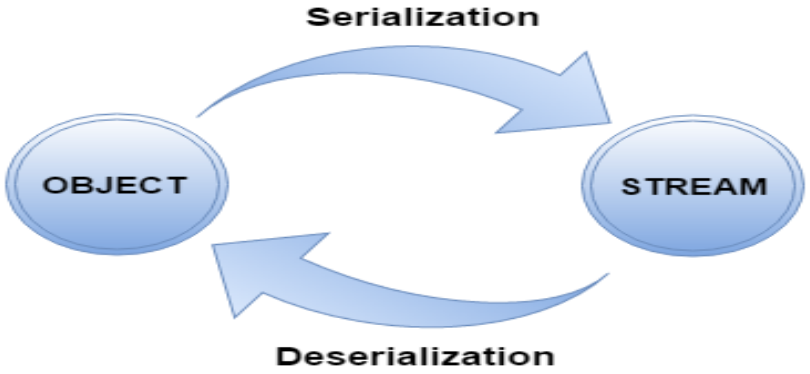


File – Data Stream – Luồng xuất dữ liệu



Xuất/Nhập dữ liệu kiểu object sử dụng luồng byte

- **Serialization và Deserialization**
  - **Serialization**: chuyển đổi trạng thái của **java object** thành **byte stream**
  - **Deserialization**: chuyển đổi trạng thái từ **byte stream** thành **java object**



- Xuất/Nhập dữ liệu kiểu object thì lớp đối tượng phải kế thừa **java.io.Serializable** interface

```java
package bkap.entity;

import java.io.Serializable;

/**
 *
 * @author QUANGND
 */
public class Student implements Serializable{

    private String studentId;
    private String studentName;

    public Student() {
    }

    public Student(String studentId, String studentName) {
        this.studentId = studentId;
        this.studentName = studentName;
    }

    public String getStudentId() {
        return studentId;
    }

    public void setStudentId(String studentId) {
        this.studentId = studentId;
    }
```

```java
public void writeObjectToFile(List<Student> listSt) {
    try {
        //Bước 1: Tạo đối tượng luồng và liên kết nguồn dữ liệu đích
        File file = new File("D:/demo.txt");
        FileOutputStream fos = new FileOutputStream(file);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        //Bước 2: Ghi đối tượng ra file - serializes object
        oos.writeObject(oos);
        //Bước 3: Đóng luồng
        fos.close();
        oos.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```java
public List<Student> readObjectFromFile() {
    List<Student> listSt = null;
    try {
        //Bước 1: Tạo đối tượng luồng và liên kết nguồn dữ liệu nguồn
        File file = new File("D:/demo.txt");
        FileInputStream fis = new FileInputStream(file);
        ObjectInputStream ois = new ObjectInputStream(fis);
        //Bước 2: Đọc đối tượng từ file - deserializes và ép kiểu về object
        listSt = (List<Student>) ois.readObject();
        //Bước 3: Đóng luồng
        fis.close();
        ois.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return listSt;
}
```

Xuất/Nhập dữ liệu sử dụng luồng character

```java
public void writeCharacterUTFToFile() {
    try {
        //Bước 1: Tạo đối tượng luồng và liên kết nguồn dữ liệu đích
        File file = new File("demo.txt");
        FileWriter fw = new FileWriter(file);
        //Bước 2: Ghi dữ liệu
        fw.write("BachKhoa-Aptech - Đơn vị đào tạo CNTT xuất sắc nhất VN");
        //Bước 3: Đóng luồng
        fw.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```java
public void readCharacterUTFFromFile() {
    try {
        //Bước 1: Tạo đối tượng luồng và liên kết nguồn dữ liệu nguồn
        File file = new File("demo.txt");
        FileReader fr = new FileReader(file);
        //Bước 2: Đọc dữ liệu
        BufferedReader br = new BufferedReader(fr);
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
        //Bước 3: Đóng luồng
        fr.close();
        br.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

HỎI ĐÁP

TRẢI NGHIỆM THỰC HÀNH

TRÂN TRỌNG CẢM ƠN!

238 Hoàng Quốc Việt, Bắc Từ Liêm, Hà Nội

0968.27.6996

tuyensinh@bachkhoa-aptech.edu.vn

www.bachkhoa-aptech.edu.vn