Day 03: 자바 고급 특성

1. **자바 주요 컬렉션**
    - `ArrayList`, `HashSet`, `HashMap` 등
2. **자바 주요 패키지**
    - `java.lang`, `java.util`, `java.io` 등
3. **JDBC**
    - DB 연결
    - CRUD 연산
4. **연습문제 (1교시)**

---

1. 자바 주요 컬렉션

자바 컬렉션 프레임워크 (Java Collection Framework) 개요

자바 컬렉션 프레임워크는 자료 구조와 알고리즘을 표준화한 인터페이스와 클래스의 집합으로, `java.util` 패키지에 포함되어 있습니다. 컬렉션 프레임워크는 데이터를 효과적으로 관리하고 조작할 수 있게 도와주는 도구들을 제공합니다.

주요 특징:

1. **표준화된 인터페이스**: List, Set, Map 등의 인터페이스를 통해 다양한 자료 구조의 표준화된 접근 방법을 제공합니다.
2. **재사용성**: 동일한 인터페이스를 구현한 여러 구현 클래스로 인해, 코드의 재사용성이 높아집니다.
3. **알고리즘 독립성**: 컬렉션에 저장된 객체들에 대해 정렬, 탐색 등의 알고리즘을 일관되게 적용할 수 있습니다.

주요 인터페이스 및 구현 클래스:

1. **List 인터페이스**: 순서가 있는 데이터의 집합. 중복 데이터를 허용합니다.
    - 주요 구현 클래스: `ArrayList`, `LinkedList`, `Vector`
2. **Set 인터페이스**: 순서가 없는 데이터의 집합. 중복 데이터를 허용하지 않습니다.

- 주요 구현 클래스: `HashSet`, `LinkedHashSet`, `TreeSet`

3. **Map 인터페이스**: 키와 값의 쌍으로 데이터를 관리합니다. 키는 중복을 허용하지 않으며, 값은 중복을 허용합니다.
   - 주요 구현 클래스: `HashMap`, `LinkedHashMap`, `TreeMap`
4. **Queue 인터페이스**: FIFO (First In, First Out) 방식으로 데이터를 관리하는 자료 구조.
   - 주요 구현 클래스: `LinkedList`, `PriorityQueue`

이외에도 여러 유틸리티 클래스와 메서드 (예: Collections.sort(), Arrays.asList() 등) 가 제공되며, 이들을 활용하면 데이터를 효율적으로 관리하고 조작할 수 있습니다.

ArrayList

정의와 특징

`ArrayList`는 자바의 컬렉션 프레임워크에 포함된 `List` 인터페이스의 구현 클래스 중 하나입니다. 그 특징은:

1. **동적 크기**: ArrayList의 크기는 자동으로 조절됩니다. 따라서 초기 크기 설정 없이 아이템을 추가하거나 제거할 수 있습니다.
2. **순서 보장**: ArrayList에 추가된 아이템들은 입력된 순서를 유지합니다.
3. **중복 허용**: 같은 아이템을 여러 번 추가할 수 있습니다.
4. **랜덤 액세스**: 인덱스를 사용하여 아이템에 빠르게 접근할 수 있습니다.

생성 및 기본 연산 (추가, 삭제, 검색 등)

```
import java.util.ArrayList;

public class ArrayListExample {

    public static void main(String[] args) {
        // ArrayList 생성
        ArrayList<String> fruits = new ArrayList<>();

        // 추가
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        System.out.println(fruits);  // [Apple, Banana, Cherry]

        // 인덱스를 사용한 추가
        fruits.add(1, "Mango");
```

```
        System.out.println(fruits);  // [Apple, Mango, Banana, Cherry]

        // 검색
        boolean hasApple = fruits.contains("Apple");  // true
        int index = fruits.indexOf("Banana");  // 2

        // 삭제
        fruits.remove("Mango");
        System.out.println(fruits);  // [Apple, Banana, Cherry]

        // 인덱스를 사용한 삭제
        fruits.remove(1);
        System.out.println(fruits);  // [Apple, Cherry]
    }
}
```

ArrayList와 일반 배열의 차이점

1. **크기**:
   - **ArrayList**: 동적으로 크기가 조절됩니다. 아이템을 추가하거나 제거할 때 자동으로 크기가 변화합니다.
   - **일반 배열**: 선언 시 고정된 크기를 가집니다. 크기를 변경하려면 새로운 배열을 만들어야 합니다.
2. **유용한 메서드**:
   - **ArrayList**: `add()`, `remove()`, `contains()`, `indexOf()` 등의 메서드를 제공하여 다양한 연산을 쉽게 수행할 수 있습니다.
   - **일반 배열**: 위와 같은 유틸리티 메서드가 내장되어 있지 않습니다. 배열 연산을 위해서는 직접 구현해야 합니다.
3. **성능**:
   - **ArrayList**: 동적 크기 조절을 위한 추가 작업이 필요하기 때문에, 때로는 일반 배열보다 느릴 수 있습니다.
   - **일반 배열**: 메모리에 연속적으로 할당되므로, 접근 속도가 빠를 수 있습니다.
4. **타입 안정성**:
   - **ArrayList**: 제네릭을 사용하여 특정 타입만 저장하도록 제한할 수 있습니다. (`ArrayList<String>`, `ArrayList<Integer>` 등)
   - **일반 배열**: 배열 선언 시 타입을 지정하여 특정 타입만 저장합니다.

HashSet

정의와 특징

`HashSet`은 자바의 컬렉션 프레임워크에 포함된 `Set` 인터페이스의 주요 구현 클래스 중 하나입니다. 그 특징은:

1. **중복 불허**: HashSet에는 중복된 아이템을 저장할 수 없습니다.
2. **순서 불보장**: HashSet에 저장된 아이템의 순서는 정해져 있지 않습니다.
3. **널 허용**: HashSet은 null 아이템을 하나 저장할 수 있습니다.
4. **해시 알고리즘**: HashSet은 내부적으로 해시 알고리즘을 사용하여 아이템을 저장 및 검색합니다.

중복 값을 허용하지 않는 이유

HashSet은 내부적으로 해시맵(`HashMap`)을 사용하여 데이터를 저장합니다. 각 아이템은 해당 아이템의 해시코드를 키로 사용하여 해시맵에 저장됩니다. 만약 동일한 아이템이 HashSet에 추가될 경우, 같은 해시코드를 가지게 되므로 기존 아이템을 덮어쓰게 됩니다. 이러한 메커니즘으로 인해 중복 아이템이 HashSet에 저장되지 않습니다.

기본 연산 (추가, 삭제, 포함 여부 확인 등)

```
import java.util.HashSet;

public class HashSetExample {

    public static void main(String[] args) {
        // HashSet 생성
        HashSet<String> animals = new HashSet<>();

        // 추가
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Bird");
        System.out.println(animals);  // [Cat, Bird, Dog] 순서는 보장되지 않습니다.

        // 중복된 아이템 추가 시도
        animals.add("Dog");
        System.out.println(animals);  // 중복된 "Dog"는 추가되지 않습니다.

        // 포함 여부 확인
        boolean hasCat = animals.contains("Cat");  // true

        // 삭제
        animals.remove("Bird");
        System.out.println(animals);  // [Cat, Dog]
```

```
        // 크기 확인
        int size = animals.size();  // 2
    }
}
```

HashMap

키와 값의 쌍으로 데이터 관리

`HashMap`은 자바의 컬렉션 프레임워크에 포함된 `Map` 인터페이스의 주요 구현 클래스 중 하나입니다. HashMap은 키(`Key`)와 값(`Value`)의 쌍으로 데이터를 저장합니다. 키는 유일해야 하므로 중복을 허용하지 않지만, 값은 중복을 허용합니다.

키의 중복을 허용하지 않는 특징

`HashMap`은 각 키에 대한 해시코드를 계산하여 데이터를 저장하거나 검색하는 데 사용합니다. 만약 중복된 키로 값을 저장하려고 시도할 경우, 해당 키의 기존 값은 새로운 값으로 대체(overwrite)됩니다. 이 특성은 `HashMap`이 키의 중복을 허용하지 않게 만드는 주된 원인입니다.

기본 연산 (값 추가, 키로 값 검색, 키/값 삭제 등)

```
import java.util.HashMap;

public class HashMapExample {

    public static void main(String[] args) {
        // HashMap 생성
        HashMap<String, Integer> studentGrades = new HashMap<>();

        // 값 추가
        studentGrades.put("Alice", 85);
        studentGrades.put("Bob", 90);
        studentGrades.put("Charlie", 78);
        System.out.println(studentGrades);  // {Alice=85, Bob=90, Charlie=78}

        // 중복된 키로 값 추가 시도
        studentGrades.put("Alice", 88);  // "Alice"의 점수를 88로 업데이트
        System.out.println(studentGrades);  // {Alice=88, Bob=90, Charlie=78}

        // 키로 값 검색
        int aliceGrade = studentGrades.get("Alice");  // 88

        // 키의 존재 여부 확인
        boolean hasBob = studentGrades.containsKey("Bob");  // true
```

```
        // 값의 존재 여부 확인
        boolean hasGrade85 = studentGrades.containsValue(85);  // false, because Alic
e's grade was updated to 88

        // 키/값 삭제
        studentGrades.remove("Bob");
        System.out.println(studentGrades);  // {Alice=88, Charlie=78}

        // 모든 키와 값을 출력
        for (String key : studentGrades.keySet()) {
            System.out.println(key + ": " + studentGrades.get(key));
        }
    }
}
```

---

2. 자바 주요 패키지

Java는 다양한 패키지를 제공하여 개발자가 일반적인 프로그래밍 작업을 더 쉽고 효율적으로 수행할 수 있도록 지원합니다. 여기에는 Java에서 가장 자주 사용되는 몇 가지 주요 패키지에 대한 간략한 개요가 포함되어 있습니다:

1. `java.lang`:
    - 이 패키지는 Java 프로그래밍 언어의 핵심 클래스들을 포함하고 있습니다.
    - `Object`, `Class`, `System`, `String`, `Thread`, `Enum` 등의 기본 클래스들이 포함되어 있습니다.
    - 기본 데이터 타입들의 Wrapper 클래스들 (`Integer`, `Character`, `Double` 등)도 포함되어 있습니다.
    - Java 프로그램에서는 이 패키지를 자동으로 import하므로 별도의 import 문 없이 사용할 수 있습니다.
2. `java.util`:
    - 유틸리티 클래스와 인터페이스를 포함하고 있으며, 자료구조 (예: `ArrayList`, `HashSet`, `HashMap` 등), 날짜 및 시간 (`Date`, `Calendar`), 이벤트 리스너, 환경 속성, 랜덤 숫자 생성 등의 기능을 제공합니다.
    - Java의 컬렉션 프레임워크 대부분이 이 패키지에 속합니다.
3. `java.io`:

- I/O (입력/출력)와 관련된 클래스와 인터페이스가 포함되어 있습니다.
- 파일 읽기/쓰기, 바이트 및 문자 스트림, 직렬화와 같은 기본적인 I/O 작업을 지원합니다.
- `File`, `FileInputStream`, `FileOutputStream`, `BufferedReader`, `PrintWriter` 등의 클래스들이 이 패키지에 포함되어 있습니다.

4. `java.net`:
   - 네트워킹 관련된 클래스와 인터페이스를 포함합니다.
   - TCP 및 UDP 소켓 프로그래밍, URL 처리 등의 기능을 제공합니다.
5. `java.sql`:
   - Java와 데이터베이스 간의 연결을 지원하는 JDBC (Java Database Connectivity) 관련 클래스와 인터페이스가 포함되어 있습니다.
   - `Connection`, `Statement`, `ResultSet` 등의 핵심 클래스들이 이 패키지에 있습니다.
6. `java.awt` 및 `javax.swing`:
   - GUI (그래픽 사용자 인터페이스) 프로그래밍을 위한 클래스와 인터페이스를 제공합니다.
   - `java.awt`는 원래의 Java GUI 컴포넌트 세트를 포함하며, `javax.swing`은 경량 컴포넌트 세트 (Swing)를 포함하고 있습니다.

이 외에도 Java는 수많은 다양한 패키지를 제공하여 다양한 프로그래밍 요구 사항을 지원합니다. 위의 개요는 Java의 주요 패키지 중 일부만 간략히 소개한 것입니다.

`java.lang`

1. 기본적인 클래스들의 집합

`Object`:

모든 Java 클래스의 최상위 클래스입니다. 여기에는 모든 객체에서 사용할 수 있는 기본 메소드들이 포함되어 있습니다.

```
Object obj = new Object();
System.out.println(obj.toString());  // java.lang.Object@<hashcode>
```

`Object` 클래스는 Java에서 모든 클래스의 최상위 클래스입니다. 이 클래스에는 몇 가지 기본 메서드가 포함되어 있습니다. 일반적으로, 사용자 정의 클래스에서는 이러한 메서드 중

일부를 오버라이드하여 클래스의 특성에 맞게 동작을 변경하곤 합니다.

실습

여기서는 `Object` 클래스의 두 가지 주요 메서드인 `toString()`과 `equals()`를 오버라이드하는 예제를 제시하겠습니다.

`Person` 클래스 정의:

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // toString() 메서드 오버라이드
    @Override
    public String toString() {
        return "Person[name=" + name + ", age=" + age + "]";
    }

    // equals() 메서드 오버라이드
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }

        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }

        Person otherPerson = (Person) obj;
        return age == otherPerson.age && (name != null ? name.equals(otherPerson.name)
: otherPerson.name == null);
    }
}
```

`Person` 클래스 사용 예제:

```
public class Test {
    public static void main(String[] args) {
        Person person1 = new Person("Alice", 25);
        Person person2 = new Person("Alice", 25);
        Person person3 = new Person("Bob", 30);

        // toString() 사용 예
```

```
        System.out.println(person1);  // Person[name=Alice, age=25]

        // equals() 사용 예
        System.out.println(person1.equals(person2));  // true
        System.out.println(person1.equals(person3));  // false
    }
}
```

위의 예제에서, 우리는 `toString()`을 오버라이드하여 `Person` 객체의 정보를 문자열로 표현하는 방법을 제공하였습니다. 또한 `equals()`를 오버라이드하여 두 `Person` 객체의 내용이 같은지 비교하는 로직을 구현하였습니다.

`String`:

문자열을 표현하는 불변(immutable) 클래스입니다. 일단 생성되면 변경할 수 없습니다.

```
String s1 = "Hello";
String s2 = " World";
String s3 = s1 + s2;
System.out.println(s3);  // Hello World
```

`String` 클래스는 문자열을 표현하고 다루는 데 사용되는 여러 유용한 메서드를 제공합니다. 여기서는 `String` 객체의 주요 메서드들을 사용하는 간단한 예제를 제공하겠습니다.

1. `length()`: 문자열의 길이를 반환합니다.

```
String str1 = "Hello";
System.out.println(str1.length());  // 출력: 5
```

2. `charAt(int index)`: 지정된 인덱스의 문자를 반환합니다.

```
System.out.println(str1.charAt(1));  // 출력: e
```

3. `substring(int beginIndex, int endIndex)`: 시작 인덱스부터 끝 인덱스 전까지의 부분 문자열을 반환합니다.

```
System.out.println(str1.substring(1, 4));  // 출력: ell
```

4. `indexOf(String substring)`: 지정된 부분 문자열의 첫 번째 발생 인덱스를 반환합니다.

```
String str2 = "Hello World";
System.out.println(str2.indexOf("World"));  // 출력: 6
```

5. `toLowerCase()` **and** `toUpperCase()`: 문자열을 소문자로 변환하거나 대문자로 변환합니다.

```
System.out.println(str2.toLowerCase());  // 출력: hello world
System.out.println(str2.toUpperCase());  // 출력: HELLO WORLD
```

6. `trim()`: 문자열의 앞뒤 공백을 제거합니다.

```
String str3 = "   Java is fun!   ";
System.out.println(str3.trim());  // 출력: Java is fun!
```

7. `replace(char oldChar, char newChar)`: 문자열에서 특정 문자를 다른 문자로 대체합니다.

```
System.out.println(str1.replace('e', 'a'));  // 출력: Hallo
```

8. `split(String regex)`: 지정된 정규 표현식을 기준으로 문자열을 분리하여 배열로 반환합니다.

```
String[] words = str2.split(" ");
for (String word : words) {
    System.out.println(word);
}
// 출력:
// Hello
// World
```

9. `equals(Object anObject)`: 두 문자열의 내용이 같은지 확인합니다.

```
String str4 = "Hello";
System.out.println(str1.equals(str4));  // 출력: true
```

10. `startsWith(String prefix)` **and** `endsWith(String suffix)`: 문자열이 특정 문자열로 시작하거나 끝나는지 확인합니다.

```
System.out.println(str2.startsWith("Hel"));  // 출력: true
```

```
System.out.println(str2.endsWith("ld"));     // 출력: true
```

이러한 메서드들은 문자열 처리 작업에서 자주 사용되며, `String` 클래스에는 이 외에도 여러 다양한 유틸리티 메서드들이 포함되어 있습니다.

`StringBuilder` :

String은 불변 객체입니다. StringBuilder나 StringBuffer는 가변적인 문자열을 표현하는 클래스입니다. 내용을 변경할 수 있으므로 문자열 조작 작업에서 성능이 좋습니다.

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");
System.out.println(sb.toString());  // Hello World
```

2. 자동으로 import되는 패키지의 특징

`java.lang` 패키지의 모든 클래스와 인터페이스는 Java 프로그램에서 기본적으로 사용할 수 있습니다. 즉, `java.lang` 패키지의 클래스를 사용하려면 별도로 `import`문을 사용할 필요가 없습니다.

예:

```
// 별도로 "import java.lang.String;"이라고 적지 않아도 String 클래스를 사용할 수 있습니다.
public class Test {
    public static void main(String[] args) {
        String message = "No need to import java.lang classes!";
        System.out.println(message);
    }
}
```

이와 같이, `java.lang` 패키지는 Java에서 가장 기본적인 클래스들을 포함하고 있으며, 이들은 개발자의 편의를 위해 자동으로 import되는 특징을 가지고 있습니다.

실습

`StringBuilder`는 가변적인 문자열을 표현하는 클래스입니다. 내용을 변경할 수 있기 때문에 문자열 조작 작업에서 높은 성능을 제공합니다. 아래는 `StringBuilder` 클래스의 주요 메서드들을 사용하는 예제입니다.

1. **생성자**

```
StringBuilder sb1 = new StringBuilder();        // 빈 문자열로 초기화
StringBuilder sb2 = new StringBuilder("Hello"); // 문자열로 초기화
```

2. `append()` : 문자열 끝에 값을 추가합니다.

```
sb1.append("Hello");
sb1.append(" World");
System.out.println(sb1);  // 출력: Hello World
```

3. `insert(int offset, String str)` : 지정된 위치에 문자열을 삽입합니다.

```
sb1.insert(5, " Java");
System.out.println(sb1);  // 출력: Hello Java World
```

4. `delete(int start, int end)` : 시작 위치와 끝 위치 사이의 문자열을 삭제합니다.

```
sb1.delete(5, 10);  // " Java" 부분을 삭제
System.out.println(sb1);  // 출력: Hello World
```

5. `reverse()` : 문자열의 순서를 반전시킵니다.

```
sb1.reverse();
System.out.println(sb1);  // 출력: dlroW olleH
```

6. `replace(int start, int end, String str)` : 시작 위치와 끝 위치 사이의 문자열을 다른 문자열로 대체합니다.

```
sb1.replace(0, 5, "World");
System.out.println(sb1);  // 출력: World olleH
```

7. `length()` : StringBuilder의 길이를 반환합니다.

```
System.out.println(sb2.length());  // 출력: 5
```

8. `charAt(int index)` : 지정된 인덱스의 문자를 반환합니다.

```
System.out.println(sb2.charAt(1));  // 출력: e
```

9. `setCharAt(int index, char ch)`: 지정된 인덱스의 문자를 변경합니다.

```
sb2.setCharAt(1, 'a');
System.out.println(sb2);  // 출력: Hallo
```

10. `toString()`: StringBuilder의 내용을 String으로 변환합니다.

```
String str = sb2.toString();
System.out.println(str);  // 출력: Hallo
```

`StringBuilder`는 문자열 조작 작업에 있어서 더 효율적인 방법을 제공합니다. 문자열의 변경 작업이 빈번할 경우 `StringBuilder`를 사용하는 것이 `String`을 직접 조작하는 것보다 더 성능에 유리합니다.

StringBuffer도 StringBuilder와 사용법이 비슷합니다.

`StringTokenizer` 클래스는 문자열을 토큰으로 분리하는 유틸리티 클래스입니다. 주어진 구분자를 기준으로 문자열을 분리하게 됩니다. 다음은 `StringTokenizer`의 주요 메서드를 사용하는 예제입니다.

StringTokenizer 사용 예제:

```
import java.util.StringTokenizer;

public class StringTokenizerExample {
    public static void main(String[] args) {
        String str = "Java,Python,C++,Ruby";

        // 쉼표(,)를 구분자로 사용하여 StringTokenizer 객체 생성
        StringTokenizer tokenizer = new StringTokenizer(str, ",");

        // 토큰이 남아 있는 동안 반복
        while (tokenizer.hasMoreTokens()) {
            System.out.println(tokenizer.nextToken());
        }
    }
}
```

출력:

```
Java
Python
C++
Ruby
```

주요 메서드:

1. `hasMoreTokens()`: 더 이상의 토큰이 있으면 `true`를 반환하고, 없으면 `false`를 반환합니다.
2. `nextToken()`: 다음 토큰을 반환합니다. 더 이상 토큰이 없을 때 호출하면 `NoSuchElementException`이 발생합니다.
3. `nextToken(String delim)`: 주어진 구분자를 사용하여 다음 토큰을 반환합니다.
4. `countTokens()`: 토큰의 총 개수를 반환합니다.

`StringTokenizer`는 구분자를 사용하여 문자열을 간단히 분리할 수 있는 방법을 제공합니다. 그러나, 최신 Java 버전에서는 `String.split()` 메서드나 `java.util.regex` 패키지의 기능을 사용하여 문자열을 분리하는 것이 더 권장됩니다.

`String.split()` 메서드 사용 예제:

`String.split()` 메서드는 주어진 정규 표현식을 사용하여 문자열을 여러 부분으로 분리합니다.

```
public class SplitExample {
    public static void main(String[] args) {
        String sentence = "Java,Python,C++,JavaScript,Rust";
        String[] languages = sentence.split(",");

        for (String lang : languages) {
            System.out.println(lang);
        }
    }
}
```

위의 코드에서는 쉼표를 기준으로 문자열을 분리하였습니다.

`java.util.regex` 패키지 사용 예제:

`Pattern`과 `Matcher` 클래스를 사용하여 문자열을 분리할 수 있습니다.

다음 코드에서는 주어진 문자열에서 숫자들만을 찾아 출력합니다.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class RegexExample {
    public static void main(String[] args) {
        String sentence = "I have 5 apples, 3 bananas, and 10 cherries.";
        String regex = "\\\\d+"; // 숫자 하나 이상을 찾는 정규 표현식

        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(sentence);

        while (matcher.find()) {
            System.out.println(matcher.group());
        }
    }
}
```

`java.util`

- 유틸리티 클래스와 인터페이스 (List, Set, Map, Collections, Date 등)

`java.util` 패키지는 자바의 핵심 패키지 중 하나로, 데이터 구조, 시간, 날짜, 기본 알고리즘 및 여러 다른 유틸리티 기능들을 위한 클래스와 인터페이스를 포함하고 있습니다. 이 패키지의 핵심 부분은 자바 컬렉션 프레임워크(JCF)로, 데이터를 저장하고 관리하는 데 사용되는 일련의 인터페이스와 구현을 제공합니다.

java.util 주요 컴포넌트:

1. **컬렉션 인터페이스**

`List` 사용 예제 (`ArrayList` 활용)

`List`: 순서대로 요소를 저장하는 컬렉션. 요소는 중복될 수 있습니다.

`List` 인터페이스는 순차적으로 요소를 저장하는 데이터 구조를 나타냅니다. `ArrayList`와 `LinkedList`는 `List` 인터페이스의 주요 구현체입니다. 여기에서는 `ArrayList`를 사용한 `List`의 예제를 제공하겠습니다.

```
import java.util.ArrayList;
import java.util.List;

public class ListExample {
    public static void main(String[] args) {
        // ArrayList 생성
        List<String> fruits = new ArrayList<>();
```

```
        // 요소 추가
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        fruits.add("Date");
        fruits.add("Elderberry");

        // 요소 출력
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // 특정 위치에 요소 추가
        fruits.add(1, "Blueberry");
        System.out.println("\\nAfter adding Blueberry at index 1:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // 요소 제거
        fruits.remove("Date");
        System.out.println("\\nAfter removing Date:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // 특정 위치의 요소 얻기
        String thirdFruit = fruits.get(2);
        System.out.println("\\nThird fruit in the list: " + thirdFruit);

        // 리스트 크기 얻기
        int size = fruits.size();
        System.out.println("\\nNumber of fruits in the list: " + size);
    }
}
```

`Set` 사용 예제 (`HashSet` 활용)

`Set`: 중복 없이 요소를 저장하는 컬렉션. (예: `HashSet`, `LinkedHashSet`, `TreeSet`)

`Set` 인터페이스는 중복 없이 요소를 저장하는 데이터 구조를 나타냅니다. `HashSet`, `LinkedHashSet`, 그리고 `TreeSet`은 `Set` 인터페이스의 주요 구현체입니다. 여기서는 `HashSet`을 사용한 `Set`의 예제를 제공하겠습니다.

```
import java.util.HashSet;
import java.util.Set;

public class SetExample {
```

```
    public static void main(String[] args) {
        // HashSet 생성
        Set<String> fruits = new HashSet<>();

        // 요소 추가
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        fruits.add("Date");
        fruits.add("Elderberry");

        // 동일한 요소를 추가하려고 시도
        fruits.add("Apple");

        // 요소 출력
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // 요소 포함 여부 확인
        if (fruits.contains("Banana")) {
            System.out.println("\\nBanana is in the set.");
        }

        // 요소 제거
        fruits.remove("Date");
        System.out.println("\\nAfter removing Date:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // Set의 크기 얻기
        int size = fruits.size();
        System.out.println("\\nNumber of fruits in the set: " + size);
    }
}
```

`Set` 사용 예제 (`HashSet` 활용)

`Map`: 키-값 쌍으로 요소를 저장하는 컬렉션. 각 키는 유일해야 합니다. (예: `HashMap`, `TreeMap`, `LinkedHashMap`)

`Set` 인터페이스는 중복 없이 요소를 저장하는 데이터 구조를 나타냅니다. `HashSet`, `LinkedHashSet`, 그리고 `TreeSet`은 `Set` 인터페이스의 주요 구현체입니다. 여기서는 `HashSet`을 사용한 `Set`의 예제를 제공하겠습니다.

```
import java.util.HashSet;
import java.util.Set;
```

```
public class SetExample {
    public static void main(String[] args) {
        // HashSet 생성
        Set<String> fruits = new HashSet<>();

        // 요소 추가
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Cherry");
        fruits.add("Date");
        fruits.add("Elderberry");

        // 동일한 요소를 추가하려고 시도
        fruits.add("Apple");

        // 요소 출력
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // 요소 포함 여부 확인
        if (fruits.contains("Banana")) {
            System.out.println("\\nBanana is in the set.");
        }

        // 요소 제거
        fruits.remove("Date");
        System.out.println("\\nAfter removing Date:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // Set의 크기 얻기
        int size = fruits.size();
        System.out.println("\\nNumber of fruits in the set: " + size);
    }
}
```

2. **유틸리티 클래스**

`Collections` 사용 예제

`Collections`: 컬렉션 관련 유틸리티 기능을 제공하는 클래스. 정렬, 검색, 동기화 작업을 수행

`Collections`는 Java에서 제공하는 유틸리티 클래스로서, 다양한 정적 메서드들을 통해 컬렉션과 관련된 연산을 지원합니다. 이 연산에는 정렬, 탐색, 변경, 동기화 등이 포함됩니다.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
```

```
public class CollectionsExample {
    public static void main(String[] args) {
        List<Integer> numbers = new ArrayList<>();

        // 요소 추가
        numbers.add(5);
        numbers.add(2);
        numbers.add(9);
        numbers.add(1);
        numbers.add(7);

        // 정렬 전 출력
        System.out.println("Before sorting: " + numbers);

        // 정렬
        Collections.sort(numbers);

        // 정렬 후 출력
        System.out.println("After sorting: " + numbers);

        // 리스트 내에서 최대값과 최소값 탐색
        int max = Collections.max(numbers);
        int min = Collections.min(numbers);

        System.out.println("Maximum value: " + max);
        System.out.println("Minimum value: " + min);

        // 특정 요소의 출현 횟수 확인
        numbers.add(5);
        numbers.add(5);
        int frequency = Collections.frequency(numbers, 5);
        System.out.println("Frequency of 5: " + frequency);

        // 리스트 반전
        Collections.reverse(numbers);
        System.out.println("After reversing: " + numbers);

        // 리스트 채우기
        Collections.fill(numbers, 0);
        System.out.println("After filling with 0: " + numbers);
    }
}
```

- `Date`: 날짜 정보를 표현하는 클래스. (최신 자바에서는 `java.time` 패키지의 클래스 권장.)
- `Random`: 난수를 생성하는 클래스.
- `TimeZone`, `Calendar`: 시간과 날짜 관련 정보를 제공하고 조작하는 클래스들.

3. **기타 클래스와 인터페이스**

- `Iterator` & `Enumeration`: 컬렉션의 요소를 순차적으로 접근하기 위한 인터페이스.

- `Properties`: 키-값 쌍의 구성 정보를 저장하고 관리하는 클래스.
- `UUID`: 유니버설 고유 식별자를 생성하는 클래스.

java.util 패키지의 중요성:

`java.util` 패키지는 자바에서 데이터 구조와 알고리즘을 다루는 데 필수적입니다. 대부분의 자바 프로젝트, 특히 자료 구조와 관련된 작업을 수행할 때 이 패키지의 클래스와 인터페이스를 활용합니다.

`java.io`

- 입력 및 출력을 위한 다양한 클래스와 인터페이스
- 파일, 네트워크, 메모리 등에서 데이터를 읽거나 쓰는 기능

`java.io` 패키지는 Java의 핵심 패키지 중 하나로, 데이터의 입력 및 출력을 위한 다양한 클래스와 인터페이스를 제공합니다. 이 패키지는 파일에서 데이터를 읽거나 쓰는 데 필요한 기능, 네트워크를 통해 데이터를 전송하는 기능, 메모리 내 데이터 스트림 처리 등의 기능을 포함하고 있습니다.

다음은 `java.io` 패키지의 주요 구성 요소와 간단한 설명입니다:

1. **Byte Streams**: 바이트 기반 스트림으로, 바이너리 데이터를 처리하는데 사용됩니다.
   - `FileInputStream` / `FileOutputStream`: 파일에서 바이트를 읽거나 쓸 때 사용됩니다.
   - `BufferedInputStream` / `BufferedOutputStream`: 버퍼링을 사용하여 효율적인 입출력을 지원합니다.
2. **Character Streams**: 문자 기반 스트림으로, 텍스트 데이터를 처리하는데 사용됩니다.
   - `FileReader` / `FileWriter`: 파일에서 문자를 읽거나 쓸 때 사용됩니다.
   - `BufferedReader` / `BufferedWriter`: 버퍼링을 사용하여 효율적인 입출력을 지원합니다.
3. **Standard Streams**: 표준 입출력 스트림. 예로, `System.in`, `System.out`, `System.err`가 있습니다.
4. **Data Streams**: 기본 데이터 타입 (int, float, long 등)을 위한 입출력 스트림.
   - `DataInputStream` / `DataOutputStream`
5. **Object Streams**: 객체의 직렬화 및 역직렬화에 사용됩니다.
   - `ObjectInputStream` / `ObjectOutputStream`

6. **Other Utilities**:
    - `PrintWriter`: 텍스트 출력을 위한 유용한 메서드를 제공합니다.
    - `Scanner`: 다양한 입력 유형을 파싱하는데 사용됩니다.

예제: 파일에서 텍스트 읽기 (`BufferedReader` 활용)

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFileExample {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new FileReader("example.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

여기서 `try-with-resources` 문을 사용하여 자원을 자동으로 닫도록 했습니다.

`java.io` 패키지에는 입출력을 위한 더 많은 클래스와 메서드가 포함되어 있습니다.

---

3. JDBC

DB 연결

- JDBC 드라이버 로딩
- Connection 객체 생성
- DB 연결 문자열 및 계정 정보 설정

Java Database Connectivity (JDBC)는 Java 응용 프로그램과 데이터베이스 사이의 표준 인터페이스를 제공합니다. Oracle Database에 연결하려면 특정한 단계를 따라야 합니다.

1. JDBC 드라이버 로딩

JDBC 드라이버는 데이터베이스와의 연결을 가능하게 하는 Java 클래스의 집합입니다. 이 드라이버는 데이터베이스 제조사 또는 제3의 제조사에서 제공될 수 있습니다.

Oracle의 경우, `ojdbc.jar`가 필요합니다. 이 JAR 파일은 Oracle Database의 설치 디렉토리에 있거나 Oracle의 공식 웹사이트에서 다운로드 할 수 있습니다.

이 드라이버를 로드하려면 다음 코드를 사용하십시오:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

이 명령은 Java의 클래스 로더를 사용하여 드라이버 클래스를 메모리에 로드합니다.

2. Connection 객체 생성

`Connection` 객체는 데이터베이스에 대한 연결을 나타냅니다. 이 객체를 통해 SQL 명령을 실행하거나 결과를 가져올 수 있습니다.

3. DB 연결 문자열 및 계정 정보 설정

데이터베이스에 연결하려면 특정한 연결 문자열이 필요합니다. 이 연결 문자열은 데이터베이스의 위치와 연결에 필요한 다른 정보를 포함합니다.

다음은 Oracle Database에 연결하는 예입니다:

```
String url = "jdbc:oracle:thin:@your_host:1521:your_sid";
String user = "your_username";
String password = "your_password";

Connection connection = DriverManager.getConnection(url, user, password);
```

- **your_host**: 데이터베이스 서버의 호스트 이름 또는 IP 주소.
- **1521**: Oracle 리스너의 기본 포트 번호. 필요에 따라 변경될 수 있습니다.
- **your_sid**: Oracle 시스템 ID (SID). 이는 Oracle 인스턴스의 고유한 이름입니다.
- **your_username**: 데이터베이스 사용자 이름.
- **your_password**: 해당 사용자의 비밀번호.

`DriverManager.getConnection()` 메서드는 주어진 연결 문자열 및 자격 증명을 사용하여 데이터베이스에 연결하려고 시도하며, 성공적으로 연결되면 `Connection` 객체를 반환합니다.

이렇게 연결된 후, 다양한 SQL 쿼리를 실행할 수 있습니다. 연결이 더 이상 필요하지 않으면, `connection.close();`를 호출하여 연결을 종료해야 합니다.

항상 필요한 라이브러리(JAR 파일)가 프로젝트의 classpath에 포함되어 있는지 확인하십시오. 이렇게 하지 않으면 연결 중 오류가 발생할 수 있습니다.

CRUD 연산

- Create: 데이터 삽입
- Read: 데이터 조회
- Update: 데이터 수정
- Delete: 데이터 삭제
- PreparedStatement 사용법 및 SQL Injection 방지

물론이죠! 아래의 요구사항을 기반으로 한 `JdbcUtil` 클래스와 간단한 `DAO` 클래스의 예제를 제시하겠습니다:

요구사항:

1. `JdbcUtil`: 데이터베이스 연결과 연결 종료를 위한 유틸리티 클래스.
2. `DAO`: 데이터 접근 객체. 데이터베이스에서 데이터를 조회, 삽입, 수정, 삭제하는 기본 기능을 가진 클래스.

```
// User.java
public class User {
    private int id;
    private String name;
    private String email;

    // Getter, Setter, Constructor, etc.
}
```

```
// JdbcUtil.java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JdbcUtil {

    private static final String JDBC_URL = "jdbc:oracle:thin:@your_host:1521:your_si
d";
    private static final String JDBC_USER = "your_username";
    private static final String JDBC_PASSWORD = "your_password";
```

```
    static {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD);
    }

    public static void closeConnection(Connection conn) {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
// SampleDAO.java
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class SampleDAO {

    public void fetchData() {
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;

        try {
            conn = JdbcUtil.getConnection();
            String sql = "SELECT * FROM your_table";
            pstmt = conn.prepareStatement(sql);
            rs = pstmt.executeQuery();

            while (rs.next()) {
                // Sample logic: Print the data
                System.out.println(rs.getString("your_column_name"));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (pstmt != null) pstmt.close();
```

```
                JdbcUtil.closeConnection(conn);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public void insertUser(User user) {
        Connection conn = null;
        PreparedStatement pstmt = null;

        try {
            conn = JdbcUtil.getConnection();
            String sql = "INSERT INTO User (id, name, email) VALUES (?, ?, ?)";
            pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, user.getId());
            pstmt.setString(2, user.getName());
            pstmt.setString(3, user.getEmail());

            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtil.closeConnection(conn);
        }
    }

    public User getUserById(int id) {
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        User user = null;

        try {
            conn = JdbcUtil.getConnection();
            String sql = "SELECT * FROM User WHERE id = ?";
            pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, id);

            rs = pstmt.executeQuery();

            if (rs.next()) {
                user = new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setEmail(rs.getString("email"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtil.closeConnection(conn);
        }

        return user;
    }
```

```
    public void updateUser(User user) {
        Connection conn = null;
        PreparedStatement pstmt = null;

        try {
            conn = JdbcUtil.getConnection();
            String sql = "UPDATE User SET name = ?, email = ? WHERE id = ?";
            pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, user.getName());
            pstmt.setString(2, user.getEmail());
            pstmt.setInt(3, user.getId());

            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtil.closeConnection(conn);
        }
    }

    public void deleteUser(int id) {
        Connection conn = null;
        PreparedStatement pstmt = null;

        try {
            conn = JdbcUtil.getConnection();
            String sql = "DELETE FROM User WHERE id = ?";
            pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, id);

            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtil.closeConnection(conn);
        }
    }
}
```

참고사항:

1. 위의 코드는 예제입니다. 실제 환경에서 실행하려면 데이터베이스 연결 정보 (`your_host`, `your_sid`, `your_username`, `your_password` 및 `your_table`, `your_column_name` 등)를 적절히 수정해야 합니다.
2. 예외 처리는 기본적인 방식으로 구현되었습니다. 실제 애플리케이션에서는 더욱 체계적인 예외 처리 전략이 필요합니다.
3. 위의 코드에서 `Connection`, `PreparedStatement`, 그리고 `ResultSet`과 같은 자원들은 사용 후 항상 닫아야 합니다. 이를 위해 `finally` 블록에서 자원을 닫는 코드를 추가하였습니다.

---

4. 연습문제

1. 컬렉션 관련 문제

1.1 ArrayList

문제: 문자열을 담는 `ArrayList`를 생성하고 5개의 문자열을 추가하십시오. 그 후, 3번째 인덱스의 문자열을 삭제하고 전체 리스트를 출력하십시오.

1.2 HashSet

문제: 정수를 담는 `HashSet`을 생성하고 1부터 10까지의 숫자 중 짝수만 추가하십시오. 그 후, 집합의 모든 원소를 출력하십시오.

1.3 HashMap

문제: 문자열 키와 정수 값으로 구성된 `HashMap`을 생성하십시오. 키는 'one', 'two', 'three' 등의 영단어이고, 값은 해당 영단어의 길이입니다. 모든 키-값 쌍을 출력하십시오.

---

2. 패키지 활용 문제

2.1 java.util

문제: `java.util` 패키지의 `Date` 클래스를 사용하여 현재 날짜와 시간을 출력하십시오. 그리고 현재 날짜로부터 10일 후의 날짜를 출력하십시오.

---

3. JDBC 실습 문제

문제: 간단한 `Users` 테이블을 생성하십시오. 이 테이블은 `id`, `name`, `email` 3개의 컬럼을 갖습니다. JDBC를 사용하여 다음 작업을 수행하십시오:

- `Users` 테이블에 새로운 사용자를 추가.
- 모든 사용자의 정보를 조회 및 출력.
- 특정 사용자의 `email` 정보를 수정.

- 특정 사용자를 삭제.