Chapter 16. 람다식

16.1 람다식이란?

- **함수형 프로그래밍(functional programming)**

함수를 정의하고 이 함수를 데이터 처리부로 보내 데이터를 처리하는 기법. 데이터 처리부는 데이터만 가지고 있을 뿐, 처리 방법이 정해져 있지 않아 **외부에서 제공된 함수에 의존**한다. 동일한 데이터라도 함수A를 제공해서 처리하는 결과와 함수B를 제공해서 처리하는 결과는 다를 수 있는데, 이것이 함수형 프로그래밍의 특징으로 **데이터 처리의 다형성**이라고도 볼 수 있다.

자바는 함수형 프로그래밍을 위해 Java 8부터 **람다식(Lambda Expressions)** 을 지원한다. 데이터 처리부는 **익명 구현 객체**로 변환된 람다식을 받아 매개변수에 데이터를 대입하고 중괄호를 실행시켜 처리한다.

인터페이스의 익명 구현 객체를 람다식으로 표현하려면 인터페이스가 **단 하나의 추상 메소드만** 가져야 한다. 인터페이스가 단 하나의 추상 메소드를 가질 때, 이를 **함수형 인터페이스(functional interface)** 라고 한다.

Calculable.java

```
package ch16.sec01;

@FunctionalInterface
public interface Calculable {
    //추상 메소드
    void calculate(int x, int y);
}
```

LambdaExample.java

```
package ch16.sec01;

public class LambdaExample {
    public static void main(String[] args) {
        action((x, y) -> {
            int result = x + y;
            System.out.println("result: " + result);
        });

        action((x, y) -> {
            int result = x - y;
            System.out.println("result: " + result);
        });
    }

    public static void action(Calculable calculable) {
        //데이터
        int x = 10;
        int y = 4;
        //데이터 처리
```

```
            calculable.calculate(x, y);
        }
    }
```

결과

```
result: 14
result: 6
```

16.2 매개변수가 없는 람다식

Workable.java

```
package ch16.sec02.exam01;

@FunctionalInterface
public interface Workable {
    void work();
}
```

Person.java

```
package ch16.sec02.exam01;

public class Person {
    public void action(Workable workable) {
        workable.work();
    }
}
```

LambdaExample.java

```
package ch16.sec02.exam01;

public class LambdaExample {
    public static void main(String[] args) {
        Person person = new Person();

        //실행문이 두 개 이상인 경우 중괄호 필요
        person.action(() -> {
            System.out.println("출근을 합니다.");
            System.out.println("프로그래밍을 합니다.");
        });
```

```
        //실행문이 한 개일 경우 중괄호 생략 가능
        person.action(() -> System.out.println("퇴근합니다."));
    }
}
```

결과

```
출근을 합니다.
프로그래밍을 합니다.
퇴근합니다.
```

16.3 매개변수가 있는 람다식

Workable.java

```
package ch16.sec03;

@FunctionalInterface
public interface Workable {
    void work(String name, String job);
}
```

Speakable.java

```
package ch16.sec03;

@FunctionalInterface
public interface Speakable {
    void speak(String content);
}
```

Person.java

```
package ch16.sec03;

public class Person {
    public void action1(Workable workable) {
        workable.work("홍길동", "프로그래밍");
    }

    public void action2(Speakable speakable) {
        speakable.speak("안녕하세요");
    }
}
```

LambdaExample.java

```
package ch16.sec03;

public class LambdaExample {
    public static void main(String[] args) {
        Person person = new Person();

        //매개변수가 두 개일 경우
        person.action1((name, job) -> {
            System.out.print(name + "이 ");
            System.out.println(job + "을 합니다.");
        });
        person.action1((name, job) -> System.out.println(name + "이 " + job + "을
하지 않습니다."));

        //매개변수가 한 개일 경우
        person.action2(word -> {
            System.out.print("\"" + word + "\"");
            System.out.println("라고 말합니다.");
        });
        person.action2(word -> System.out.println("\"" + word + "\"라고 외칩니
다."));
    }
}
```

결과

```
홍길동이 프로그래밍을 합니다.
홍길동이 프로그래밍을 하지 않습니다.
"안녕하세요"라고 말합니다.
"안녕하세요"라고 외칩니다.
```

16.4 리턴값이 있는 람다식

Calcuable.java

```
package ch16.sec04;

@FunctionalInterface
public interface Calcuable {
    double calc(double x, double y);
}
```

Person.java

```
package ch16.sec04;

public class Person {
    public void action(Calcuable calcuable) {
        double result = calcuable.calc(10, 4);
        System.out.println("결과: " + result);
    }
}
```

LambdaExample.java

```
package ch16.sec04;

public class LambdaExample {
    public static void main(String[] args) {
        Person person = new Person();

        //실행문이 두 개 이상일 경우
        person.action((x, y) -> {
            double result = x + y;
            return result;
        });

        //리턴문이 하나만 있을 경우(연산식)
        //person.action((x, y) -> {
        // return (x + y);
        //});
        person.action((x, y) -> (x + y));

        //리턴문이 하나만 있을 경우(메소드 호출)
        //person.action((x, y) -> {
        // return sum(x, y);
        //});
        person.action((x, y) -> sum(x, y));
        }

    public static double sum(double x, double y) {
        return (x + y);
    }
}
```

결과

```
결과: 14.0
결과: 14.0
결과: 14.0
```

16.5 메소드 참조

정적 메소드와 인스턴스 메소드 참조

Calcuable.java

```
package ch16.sec05.exam01;

@FunctionalInterface
public interface Calcuable {
    double calc(double x, double y);
}
```

Person.java

```
package ch16.sec05.exam01;

public class Person {
    public void action(Calcuable calcuable) {
        double result = calcuable.calc(10, 4);
        System.out.println("결과: " + result);
    }
}
```

Computer.java

```
package ch16.sec05.exam01;

public class Computer {
    public static double staticMethod(double x, double y) {
        return x + y;
    }

    public double instanceMethod(double x, double y) {
        return x * y;
    }
}
```

MethodReferenceExample.java

```
package ch16.sec05.exam01;

public class MethodReferenceExample {
```

```
    public static void main(String[] args) {
        Person person = new Person();

        //정적 메소드일 경우
        //람다식
        //person.action((x, y) -> Computer.staticMethod(x, y));
        //메소드 참조
        person.action(Computer :: staticMethod);

        //인스턴스 메소드일 경우
        Computer com = new Computer();
        //람다식
        //person.action((x, y) -> com.instanceMethod(x, y));
        //메소드 참조
        person.action(com :: instanceMethod);
    }
}
```

결과

```
결과: 14.0
결과: 40.0
```

매개변수의 메소드 참조

Comparable.java

```
package ch16.sec05.exam02;

@FunctionalInterface
public interface Comparable {
    int compare(String a, String b);
}
```

Person.java

```
package ch16.sec05.exam02;

public class Person {
    public void ordering(Comparable comparable) {
        String a = "홍길동";
        String b = "김길동";

        int result = comparable.compare(a, b);

        if(result < 0) {
            System.out.println(a + "은 " + b + "보다 앞에 옵니다.");
```

```
        } else if(result == 0) {
            System.out.println(a + "은 " + b + "과 같습니다.");
        } else {
            System.out.println(a + "은 " + b + "보다 뒤에 옵니다.");
        }
    }
}
```

MethodReferenceExample.java

```
package ch16.sec05.exam02;

public class MethodReferenceExample {
    public static void main(String[] args) {
        Person person = new Person();
        person.ordering(String :: compareToIgnoreCase);
    }
}
```

결과

```
홍길동은 김길동보다 뒤에 옵니다.
```

16.6 생성자 참조

Creatable1.java

```
package ch16.sec05.exam03;

@FunctionalInterface
public interface Creatable1 {
    public Member create(String id);
}
```

Creatable2.java

```
package ch16.sec05.exam03;

@FunctionalInterface
public interface Creatable2 {
    public Member create(String id, String name);
}
```

Member.java

```
package ch16.sec05.exam03;

public class Member {
    private String id;
    private String name;

    public Member(String id) {
        this.id = id;
        System.out.println("Member(String id)");
    }

    public Member(String id, String name) {
        this.id = id;
        this.name = name;
        System.out.println("Member(String id, String name)");
    }

    @Override
    public String toString() {
        String info = "{ id: " + id + ", name: " + name + " }";
        return info;
    }
}
```

Person.java

```
package ch16.sec05.exam03;

public class Person {
    public Member getMember1(Creatable1 creatable) {
        String id = "winter";
        Member member = creatable.create(id);
        return member;
    }

    public Member getMember2(Creatable2 creatable) {
        String id = "winter";
        String name = "한겨울";
        Member member = creatable.create(id, name);
        return member;
    }
}
```

ConstructorReferenceExample.java

```java
package ch16.sec05.exam03;

public class ConstructorReferenceExample {
    public static void main(String[] args) {
        Person person = new Person();

        Member m1 = person.getMember1( Member :: new );
        System.out.println(m1);
        System.out.println();

        Member m2 = person.getMember2( Member :: new );
        System.out.println(m2);
    }
}
```

결과

```
Member(String id)
{ id: winter, name: null }

Member(String id, String name)
{ id: winter, name: 한겨울 }
```