일자 : 2019,01.16 20:00 ~ 22:00 (120분)

참석자 : 김경민, 박지환, 임현철

1. 3장 인터페이스와 람다 표현식
   1. 인터페이스
      * 서비스 공급자와 자신의 객체를 이 서비스에 사용하고 싶은 클래스 간의 계약을 기술하는 메커니즘이다.
      * 모든 메서드는 자동으로 public이 된다. 하지만 의도를 더 명확하게 드러내기위해 public으로 직접 선언 하기도 한다.
      * 인터페이스를 구현하는 클래스는 인터페이스의 메서드를 반드시 public으로 선언해야한다.
      * 인터페이스 타입으로 변수를 선언할 수 있지만, 타입이 인터페이스 자체인 객체는 만들 수 없다. 모든 객체는 클래스의 인스턴스이다.
      * Instanceof 연산자는 null에 안전하다. (obj가 null이면 instanceof는 false를 반환한다.)
      * 인터페이스 안에는 인스턴스 변수를 둘 수 없다. 인터페이스는 객체의 상태가 아니라 동작을 명시한다.

|  |
| --- |
| public interface IntSequence {  boolean hasNext();  int next();  } |

* 1. 인터페이스의 정적 메서드, 기본 메서드, 비공개 메서드
     + 자바8 이전에는 인터페이스의 모든 메서드가 추상 메서드여야 했다. 하지만 자바9 에서는 실제 구현이 있는 메서드 세 종류(정적 메서드, 기본메서드, 비공개 메서드)를 인터페이스에 추가할 수 있다.
     + 다시말해, 자바8 부터는 정적 메서드와 기본 메서드를 인터페이스에 넣을 수 있고, 자바9 부터는 비공개 메서드를 인터페이스에 넣을 수 있다.
     + 정적메서드는 인터페이스를 구현한 클래스의 인스턴스를 돌려주지만, 호출자는 이 인스턴스가 어느 클래스의 인스턴스인지 신경 쓸 필요가 없다.
     + 기본 메서드는 인터페이스에 있는 어느 메서드에서든 기본(default) 구현을 작성할 수 있다.

|  |
| --- |
| public interface Identified {  default int getId() { return Math.abs(hashCode()); }  } |

* + - 기본 메서드의 충돌시 super 키워드로 슈퍼타입의 메서드를 호출하여 어느 슈퍼타입을 원하는지 명시하여 해결할 수 있다.
    - 비공개 메서드는 static이나 인스턴스 메서드는 될 수 있지만, 오버라이드가 가능하므로 default 메서드는 될 수 없다. 인터페이스 자체에 있는 메서드에서만 쓸 수 있으므로, 인터페이스 안에 있는 다른 메서드의 헬퍼 메서드로만 사용할 수 있다.
  1. 람다 표현식
     + 나중에 한 번 이상 실행할 수 있게 전달하는 코드 블록이다.

|  |
| --- |
| public class LambdaDemo {  public static void main(String[] args) {  String[] friends = { "Peter", "Paul", "Mary" };  Arrays.sort(friends,  (first, second) -> first.length() - second.length());  System.out.println(Arrays.toString(friends));  ArrayList<String> enemies = new ArrayList<>(Arrays.asList("Malfoy", "Crabbe", "Goyle", null));  enemies.removeIf(e -> e == null);  System.out.println(enemies);  }  } |

* 1. 함수형 인터페이스
     + 함수형 프로그래밍 언어는 대부분 함수 타입이 구조적(structural)이다.
     + 대부분의 표준 함수형 인터페이스에는 함수를 생성하거나 결합하는 비추상 메서드가 있다.
  2. 지역 클래스와 익명 클래스
     + 지역클래스(로컬 클래스)는 메서드 안에 정의한 클래스이다.

|  |
| --- |
| public class LocalClassDemo {  private static Random generator = new Random();  public static IntSequence randomInts(int low, int high) {  class RandomSequence implements IntSequence {  public int next() { return low + generator.nextInt(high - low + 1); }  public boolean hasNext() { return true; }  }  return new RandomSequence();  }  public static void main(String[] args) {  IntSequence dieTosses = randomInts(1, 6);  for (int i = 0; i < 10; i++) System.out.println(dieTosses.next());  }  } |

* + - 익명 클래스로도 정의할 수 있다..

|  |
| --- |
| public class AnonymousClassDemo {  private static Random generator = new Random();  public static IntSequence randomInts(int low, int high) {  return new IntSequence() {  public int next() { return low + generator.nextInt(high - low + 1); }  public boolean hasNext() { return true; }  };  }  public static void main(String[] args) {  IntSequence dieTosses = randomInts(1, 6);  for (int i = 0; i < 10; i++) System.out.println(dieTosses.next());  }  } |

* 1. 핵심 내용 정리
     + 인터페이스는 구현 클래스에서 반드시 구현해야 하는 메서드를 명시한다.
     + 인터페이스는 해당 인터페이스를 구현하는 모든 클래스의 슈퍼타입이다. 따라서 구현 클래스의 인스턴스를 인터페이스 타입 변수에 할당 할 수 있다.
     + 인터페이스는 정적 메서드를 포함할 수 있다. 인터페이스의 모든 변수는 자동으로 public static final이다.
     + 인터페이스는 구현 클래스에서 상속하거나 오버라이드할 수 있는 기본 메서드를 포함할 수 있다.
     + 인터페이스는 구현 클래스에서 호출하거나 오버라이드할 수 없는 비공개 메서드를 포함할 수 있다.
     + Comparable과 Comparator 인터페이스는 객체를 비교할 때 사용한다.
     + 함수형 인터페이스는 단일 추상 메서드를 가진 인터페이스다.
     + 람다 표현식은 나중에 실행할 수 있는 코드 블록이다.
     + 람다 표현식은 함수형 인터페이스로 변환된다.
     + 메서드 참조와 생성자 참조는 메서드와 생성자를 호출하지 않고 참조한다.
     + 람다 표현식과 지역 클랙스는 자신을 감싸는 유효 범위에 있는 사실상 최종 변수에 접근할 수 있다.

1. 4장 상속과 리플렉션
   1. 클래스 확장
      * Extends 키워드는 기존 클래스에서 파생된 새 클래스를 만든다는 것을 나타낸다. 이때 기존 클래스를 슈퍼클래스, 새 클래스를 서브클래스라고 한다.

|  |
| --- |
| Public class Manager extends Employee{    } |

* 1. Object: 보편적 슈퍼클래스

|  |  |
| --- | --- |
| **메서드** | **설명** |
| String toString() | 객체의 문자열 표현을 돌려준다. 기본 값은 클래스 이름과 해시 코드다(예를 들어 java.lang.Object@3c407114) |
| Boolean equals(Object other) | 객체가 other와 같으면 true를 반환하고, 다르거나 other가 null이면 false를 반환한다. 기본 값은 두 객체가 동일해야 같다고 취급한다. obj.equals(other)보다는 null에 안전한 Object.equals(obj, other)를 사용한다. |
| Int hashCode() | 객체에 해당하는 해시 코드를 돌려준다. 같은 객체면 해시 코드도 같아야한다. 오버라이드하지 않으면 가상 머신이 특정한 방법으로 해시 코드를 할당한다. |
| Class<?> getClass() | 객체가 속한 클래스를 기술하는 Class 객체를 돌려준다. |
| Protected Object clone() | 객체를 복제한다. 기본 값은 얕은 복사다. |
| Protected void finalize() | 가비지 컬렉터가 객체를 회수할 때 호출되는 메서드다. 오버라이드하지 말아야한다. |
| wait, notify, notifyAll | 병행프로그래밍 |

* 1. 열거
     + 열거 타입의 각 인스턴스는 유일하다.
     + 열거 타입에 생성자, 메서드, 필드를 추가하는 것도 가능하다.

|  |
| --- |
| public enum Size {  SMALL("S"), MEDIUM("M"), LARGE("L"), EXTRA\_LARGE("XL");  private String abbreviation;  Size(String abbreviation) {  this.abbreviation = abbreviation;  }  public String getAbbreviation() { return abbreviation; }  } |

* + - Enum 인스턴스 각각에 메서드를 추가할 수 있지만, 열거에 정의된 메서드를 오버라이드 하는 것이어야 한다. 예를 들어 계산기를 구현한다면 아래와 같다.

|  |
| --- |
| public enum Operation {  ADD("+") {  public int eval(int arg1, int arg2) { return arg1 + arg2; }  },  SUBTRACT("-") {  public int eval(int arg1, int arg2) { return arg1 - arg2; }  },  MULTIPLY("\*") {  public int eval(int arg1, int arg2) { return arg1 \* arg2; }  },  DIVIDE("/") {  public int eval(int arg1, int arg2) { return arg1 / arg2; }  };  private String symbol;  Operation(String symbol) { this.symbol = symbol; }  public String getSymbol() { return symbol; }    public abstract int eval(int arg1, int arg2);  } |

* 1. 실행 시간 타입 정보와 리소스
     + 리소스로드, Class가 제공하는 유용한 서비스 중 하나는 설정 파일이나 이미지처럼 프로그램에 필요한 리소스를 찾아오는 것이다. 클래스 파일이 있는 곳과 같은 디렉터리에 리소스를 넣었다면 다음 방법으로 리소스 파일에 대응하는 입력 스트림을 열 수 있다.

|  |
| --- |
| InputStream stream = MyClass.class.getResourceAsStream(“config.txt”);  Scanner in = new Scanner(stream); |

* + - 클래스로더, 클래스 파일에는 가상 머신 명령어가 저장된다. 각 클래스 파일은 단일 클래스나 인터페이스에 해당하는 명령어를 담는다. 클래스 파일을 파일시스템, Jar파일, 원격 위치에 둘수있고, 심지어 메모리에서 동적으로 생성할 수도 있다. 클래스로더는 바이트를 로드해서 가상머신의 클래스나 인터페이스로 변환하는 역할을 한다.
    - 부트스트랩 클래스 로더는 가장 기본적인 자바 라이브러리 클래스를 로드한다. 이로더는 가상머신의 일부다.
    - 플랫폼 클래스 로더는 다른 라이브러리 클래스를 로드한다. 부트스트랩 클래스 로더가 로드하는 클래스와 달리 보안 정책으로 플랫폼 클래스 퍼미션을 구성할 수 있다.
    - 시스템 클래스 로더는 애플리케이션 클래스를 로드한다. 이 로더는 클래스 패스와 모듈 패스에 있는 디렉터리와 Jar 파일에서 클래스를 찾는다.
  1. 리플렉션

프로그램에서 리플렉션을 이용하면 실행 시간에 객체 내용을 조사하고, 해당 객체에 있는 임의의 메서드를 호출할 수 있다. 리플렉션은 객체-관계 매퍼나 GUI 빌러 같은 도구를 구현할 때 유용하다.

* 1. 핵심 내용 정리
     + 서브클래스는 슈퍼클래스에서 private이 아닌 메서드를 상송하거나 오버라이드할 수 있다.
     + Super 키워드로 슈퍼클래스의 메서드나 생성자를 호출할 수 있다.
     + Final 메서드는 오버라이드할 수 없고, final 클래스는 확장할 수 없다.
     + Abstract 메서드는 구현이 없고, abstract 클래스의 인스턴스는 생성할 수 없다.
     + 슈퍼클래스의 protected 멤버는 서브클래스 메서드에서 접근할 수 있다. 하지만 같은 서브클래스의 객체에 적용할 때믄 가능하다. Protected 멤버는 패키지 내부에서도 접근할 수 있다.
     + 클래스는 모드 Object의 서브클래스다. Object 클래스에는 toString, equals, hashCode, clone 메서드가 있다.
     + 열거 타입은 Enum의 서브클래스다. Enum 클래스에는 인스턴스 메서드 toString, compareTo 등과 정적 메서드 valueOf가 있다.
     + Class 클래스는 자바 타입과 관련된 정보를 제공한다. 정보를 제공할 수 있는 자바 타입은 클래스, 배열, 인터페이스, 기본 타입, void다.
     + Class 객체를 사용하면 클래스 파일과 함께 있는 리소스를 로드할 수 있다.
     + 클래스 로더를 사용하면 클래스 패스에 지정되지 않은 위치에서 클래스를 로드할 수 있다.
     + ServiceLoader 클래스는 서비스 구현체를 찾아 선택하는 메커니즘을 제공한다.
     + 리플렉션 라이브러리를 사용하면 프로그램에서 객체의 멤버를 발견하고, 변수에 접근하고, 메서드를 호출할 수 있다.
     + 프록시 객체는 임의의 인터페이스를 동적으로 구현하며, 모든 메서드 호출을 핸들러로 전달한다.