8/22

▼ 어제 수업 복습 🤓

1. 변수와 데이터 타입

- **기본 데이터 타입 (Primitive Type)**
  - 변수 안에 값 자체가 저장된다.
  - 예: `int`, `double`, `char`, `boolean` 등
- **참조 타입 (Reference Type)**
  - 값이 저장된 메모리의 **주소값**을 변수에 저장한다.
  - 실제 데이터는 **Heap 영역**에 존재한다.
  - 예: `String`, 배열, 클래스 등

👉 변수는 해당 타입의 값 하나만 저장 가능하다.

---

2. 배열 (Array)

- 여러 개의 값을 하나의 변수처럼 관리할 수 있다.
- **인덱스(index)** 를 이용해 각 값에 접근한다.
- 종류
  - 1차원 배열: 단순한 리스트 형태
  - 2차원 배열: 행과 열 구조 (테이블 형태)
  - 3차원 배열 이상: 잘 사용하지 않지만 가능하다

예:

```
int[] numbers = {1, 2, 3, 4};
System.out.println(numbers[2]); // 3 출력
```

---

3. 메소드 (Method)

- **객체가 가진 기능(동작)을 표현하는 단위**
- **매개변수 (Parameter)** 를 통해 외부 값 전달 가능
- **인자 (Argument)**: 메소드를 호출할 때 전달하는 실제 값
  - 예: `add(20, 31)` → `20`, `31`이 인자

특징

1. **메소드 오버로딩 (Overloading)**
   - 같은 이름의 메소드를 여러 개 정의할 수 있다.
   - 단, 매개변수의 타입이나 개수가 달라야 함.
2. **가변 길이 매개변수 (Varargs)**
   - 개수가 정해지지 않은 인자를 받을 수 있다.
   - 문법: `int... numbers`
   - 예:

```
int sum(int... numbers) {
    int total = 0;
    for(int n : numbers) total += n;
    return total;
}

sum(10, 20, 30, 40); // 인자 4개 전달 가능
```

---

4. 객체지향 프로그래밍 (OOP)

- **객체**: 스스로 역할과 책임을 가진 독립적인 단위
- **추상화 (Abstraction)**
  - 불필요한 부분은 제거하고, 본질적이고 필요한 것만 남기는 과정
  - 자바의 클래스, API 라이브러리 모두 추상화의 결과물

클래스와 인스턴스

- **클래스 (Class)**: 객체를 만들기 위한 설계도

- **인스턴스화 (Instantiation)**: 클래스로부터 실제 객체(인스턴스)를 생성하는 과정
  - `new` 키워드 사용
  - 비유: "클래스 = 건물 도면, 인스턴스 = 실제 건물"

📌 접근 제한자 정의

- **public**
  - 어디서든 접근 가능 (같은 클래스, 같은 패키지, 다른 패키지 모두)
- **protected**
  - 같은 패키지 내에서는 접근 가능
  - 다른 패키지라도 **상속 관계**라면 접근 가능
- **(default, 아무것도 안 쓴 경우)**
  - 같은 패키지 내에서만 접근 가능
- **private**
  - 같은 클래스 내에서만 접근 가능

👉 정리하자면, 공개 범위가 **넓은 순서 → public > protected > default > private**

---

5. static 키워드

- *static 멤버(필드/메소드)**는 Heap이 아닌 **메소드 영역(Method Area)** 에 저장된다.
- 인스턴스를 생성하지 않아도 접근 가능하다.
- 모든 객체가 **하나의 static 필드를 공유**한다.
- 활용 예:
  - `Math.random()`, `Math.pow()`
  - `static int count;` → 객체 수를 셀 때 유용

---

1. Dice 예제 정리

- `Dice` 클래스 → **객체 설계도**

- 필드: `face` , `eye`
- 메소드: `roll()` (주사위를 굴려 무작위 값을 저장)

- `DiceUser` 클래스 → **객체 활용 코드**
  - 객체 생성 : `Dice dice = new Dice();` `// new 키워드 + 생성자 호출`
  - 객체 사용 : `dice.roll(); //메소드 실행` , `System.out.println(dice.eye);`
  - 10번 굴려 3의 빈도수, 100번 굴려 5의 빈도수를 확인

```
//class 정의
import java.util.Random;

public class Dice {
	//필드(멤버변수) 몇개의면, 사이즈, 색상
	int face = 6;
	int eye;

	//메소드 주사위 굴리기, 주사위 값 확인
	public void roll () {
	   //주사위의 면의 따라서 랜덤한 값을 발생
	   Random random = new Random();

	   //0.0 ~ 1.0 사이 값을 리턴 (1.0은 포함이 안됨)
	   //Math.random();
	   eye = (int)(Math.random()*face)+1;

	}

   public static void main(String[] args) {
	   //Math.random()*6 ⇒ 0-5까지 출력됨
	System.out.println((int)(Math.random()*6)+1);
   }
}
```

```
//객체 생성+사용
public class DiceUser {
   public static void main(String[] args) {
	Dice dice = new Dice();
```

```
        dice.roll(); //주사위를 굴림
        int count = 0; //카운트

        System.out.println("주사위를 굴려서 나온 눈 : " + dice.eye);


        // 주사위를 10번 굴려서 3이 몇번 나오는지
            for(int i = 0; i < 10; i++){
                dice.roll();
                System.out.print(dice.eye + "\t");
                if(dice.eye == 3) count++;
            }
            System.out.println("\n주사위를 10번 굴려 3이 나온 횟수 : " + count );

        //주사위를 100번 굴려서 5번이 나오는지
        for(int i = 0; i < 100; i++) {
            dice.roll();
            if(dice.eye == 5) count++;
        }
        System.out.println("주사위를 100번 굴려 5번이 나온 횟수 " + count);

    }
}
```

---

2. 리팩토링(Refactoring)

- **정의**: 기능은 그대로 유지하면서, 가독성·재사용성·효율성을 높이도록 코드 구조를 개선하는 과정.
- **Dice 예제에서 리팩토링 아이디어**
  - 중복 코드 제거 (예: 주사위를 여러 번 굴려 특정 값이 나오는 횟수를 세는 부분 → 메소드로 분리 가능)
  - `count` 변수의 재사용 대신, 독립적인 메소드로 관리하면 더 명확해짐.
  - `Random random = new Random();` → 클래스 필드로 빼서 한 번만 생성 가능.

3. 메소드 오버로딩 (Method Overloading)

- **정의**

  같은 클래스 안에서 **메소드 이름은 같지만 매개변수의 개수나 타입이 다른 메소드**를 여러 개 정의하는 것.

  - **중요:** 반환 타입만 다른 것은 오버로딩이 아님.

```
int sum(int a, int b) { ... }
double sum(double a, double b) { ... }  // 가능
int sum(int x, int y) { ... }           // 불가능, 이미 같은 매개변수
```

- **오버로딩 조건**

  1. 메소드 이름은 동일해야 함
  2. 매개변수 **개수** 또는 **타입**이 달라야 함
  3. 반환 타입만 다른 것은 오버로딩으로 인정되지 않음

- **목적 / 이유**

  1. **코드의 가독성 향상:** 같은 기능을 수행하는 메소드에 여러 이름을 붙일 필요 없이 하나의 이름으로 통일 가능
  2. **사용자 편의성:** 사용자는 매번 다른 메소드 이름을 외울 필요 없이 같은 이름으로 다양한 입력에 대응 가능
  3. **유지보수 용이:** 기능 변경 시 하나의 이름으로 관리 가능

- **오버라이딩이란?**

  1. 상속받은 메서드를 자식 클래스에서 재정의하여 다른 동작을 하도록 만드는 것

---

4. 생성자(Constructor)

- **정의**: 클래스 이름과 동일하며 리턴 타입이 없는 특별한 메소드.

  → 객체가 생성될 때 자동으로 호출되어 **초기 상태를 설정**함.

- **주요 역할**:

1. 객체 생성 시 **필드 초기화**
2. 객체 생성과 동시에 필요한 **초기 작업 수행**

- **특징**:
  - 리턴 타입 명시를 하지 않는다. (`void` 포함 X)
  - 이름이 **클래스 이름과 동일**해야 함
  - **매개변수 유무 가능**
  - **객체 생성 시 단 한 번만 호출**되며, 임의 호출 불가
- **기본 생성자(Default Constructor)**:
  - 클래스에 생성자를 정의하지 않으면, 자바가 **자동으로 매개변수 없는 생성자**를 제공
  - 따라서 `new Dice()`처럼 매개변수가 없는 생성자 호출 가능
- **주의 사항**:
  - 객체 생성 시 초기값이 없으면 null 또는 0으로 설정됨 → 예: 회원 객체에서 이름·성별·나이가 없으면 **유령 회원** 발생 가능
- **예**

```
Person person = new Person(); // 생성자 호출
```

4-1. this

- **정의**: 현재 객체 자신을 가리키는 **특별한 참조 변수**
- **사용 위치**: 클래스 내부, **메소드 또는 생성자 안**
- **주요 역할**:
  1. **멤버 변수와 매개변수 구분**

     ```
     this.name = name; // 필드와 매개변수 이름이 같을 때
     ```

  2. **같은 클래스 내 다른 생성자 호출**

     ```
     this(...); // 반드시 생성자 첫 줄에 작성
     ```

3. **현재 객체 참조 반환**
    - 메소드 체인 등에서 활용 가능

- **주의 사항**:
    - `this()`는 **생성자 코드의 첫 줄**에서만 사용 가능

      → 객체를 일관되게 초기화하고 **초기화 순서 보장**을 위해
    - 생성자와 매개변수 이름이 같을 때, **'this'로 명확히 구분**

---

5. Math & Random

5-1. Math 클래스

- **정의**: 수학 연산과 상수, 간단한 난수를 제공하는 클래스 (`java.lang`)
- **특징**: 객체 생성 없이 `Math.메소드()`로 바로 사용 가능
- **난수 활용**: `Math.random()`은 0.0 이상 1.0 미만의 double 난수 반환
- 예제

```
int dice = (int)(Math.random() * 6) + 1; // 1~6 난수
System.out.println(dice);
```

원하는 범위의 정수 난수는 (int)(Math.random()*범위)+시작값 방식으로 변환

난수 : 예**측할 수 없는 수** 또는 **임의로 결정된 수 (게임, 시뮬레이션, 암호화, 통계** 등에서 활용)

메소드 체이닝 : 객체가 메소드를 실행한 뒤 자기자신을 반환하여 여러 메소드를 연속적으로 한 줄에 호출

5-2. Random 클래스

- **정의**: 난수 생성을 위해 제공되는 클래스 (`java.util.Random`)
- **특징**: 다양한 타입의 난수를 생성 가능, 시드(seed)를 주면 재현 가능

- **사용 방법**: 객체 생성 후 메소드 호출
- 예제

```
import java.util.Random;

Random rand = new Random();
int dice = rand.nextInt(6) + 1; // 1~6 난수
System.out.println(dice);
```

5-3. 차이점 정리

구분	Math.random()	Random 클래스
사용법	static 메소드, 객체 필요 없음	객체 생성 후 메소드 사용
난수 범위	0.0~1.0 (double)	int, double, boolean 등 다양한 타입

5-4. 실제 코드 활용 예시 (주사위 굴리기)

```
Dice dice = new Dice();
dice.roll(); // 주사위를 굴림
System.out.println("주사위 눈: " + dice.eye);

// 10번 굴려서 3 나온 횟수
int count = 0;
for(int i = 0; i < 10; i++){
    dice.roll();
    if(dice.eye == 3) count++;
}
System.out.println("3 나온 횟수: " + count);
```

6. 패키지(Package)

- **정의**: 관련 클래스를 그룹화하여 관리하는 단위

  → **폴더처럼 클래스들을 정리**하고, **클래스 이름 충돌 방지**

- **사용법**: `import`를 통해 다른 패키지의 클래스 사용 가능
- **패키지 이름 규칙**:
  - 모두 **소문자** 사용
  - **도메인 역순** 사용 (예: `com.example.project`)
  - **예약어 사용 금지**
  - **특수문자**는 언더스코어(`_`)만 허용

개인회고

> 남혜린
>
> 오늘은 써머리를 정리하는 날이라 집중이 잘 되었고, 정리를 하면서 이해도 한층 더 깊어졌다. 생성자 부분부터는 조금 벅차서 지치기도 했지만, 되짚어보며 정리하는 과정에서 얻은 점이 많아 복기에 큰 도움이 될 것 같다. 또한 팀원들과 함께 피드백하고 수정해가며 완성도가 높아지는 과정을 보며 협업의 의미도 다시 느낄 수 있었다. 형규님 말씀처럼 어제의 나보다 오늘의 내가 더 나아져야 한다는 다짐을 다시금 새기게 되었고, 주말 동안 이번 주 진도를 복습해 다음 주 수업을 무리 없이 따라갈 수 있도록 최선을 다하겠다. 😤

> 백종현
>
> 오늘은 본격적인 객체지향 개념을 배웠습니다.
> 오늘 문제를 풀며 팀원들분에게 어떻게 설명 해야 할 까 라는 생각으로 문제를 풀고 학습하니까 도움이 많이 되었던것 같습니다. 객체지향은 정말 어렵고 애매모호 하지만 중요한 개념인것 같습니다.
> 다같이 객체자향을 완벽하게 정복하길 기원하며 오늘의 회고를 마치겠습니다!
>
> - 객체

- 추상화
- 생성자
- 메모리
- heap, stack
- 오버로딩

윤수정

오늘 부트캠프에서 배운 내용이 정말 흥미로웠다. 본격적으로 객체지향 개념을 배우기 시작했는데, 특히 추상화 부분은 아직 감이 잘 오지 않아 더 많은 연습이 필요하다고 느꼈다.

이번 주말에는 강사님이 내주신 실습문제와 교안에 있는 실습문제들을 모두 풀어보고, 로또 당첨기 프로그램도 직접 구현해볼 예정이다. 실습을 통해 감각을 익히는 시간이 많이 필요할 것 같다.

또한 회고 시간에 이번 달 말에 만나자는 약속이 잡혔지만, 일정이 맞지 않아 참석하지 못하게 된 점이 아쉽다. 다음 달에는 꼭 대면으로 참여할 수 있기를 바란다. 더 친밀해지는 우리팀이 되기를!!

최형규

오늘까지 객체지향 기초 까지 여러 개념들을 익히고, 주사위, 계산기, 로또 등 실습문제를 부지런히 따라갔다.

직접 코드쓰는게 아직 어려웠지만, 지피티와 다른분들 하신것들 보면서 따라갔다

그렇다 따라가기만 했다. 그래도 중간중간 복습하거나 다시 읽어볼 시간이 있어서

어떻게 공부해야하고 내가 모르는부분이 어떤것인지 알게되어 주말에 이번 주 익힌것만이라도 잘 소화시켜야겠다.

오늘 김교수님 외근 가셔서 아쉬웠지만 우리에겐 백교수님이 있었다. ㅋㅋㅋㅋ

아무튼 교수님들도 멘티들도 좀 성장곡선 가파르게 찍어서

같이 프로젝트도 해보고 재밌게 하구 싶다.

또 스케줄잡기가 녹록치 않은데, 혹 같이 없더라도 같이 있는것처럼 즐거운 팀플 되면 좋겠다.

주말에 정말 질문거리 엄청 생길 것 같은데, 일딴..

주3시간 운동, 이번주 한거 다 소화시키기 무조건 성공시키자

이환진

오늘 수업은 진도를 깊게 나가지않았다.

객체 지향 기본까지 나갔고, 생성자, 스태틱 등 배웠다. 느낀점은 솔직히 처음배워서 아직 제대로 이해를 못하고 있다.

내가 해야할 일은 복습을 꾸준히해서

이해할수있을때까지 해야겠다.

만약 내가 이해를 했다면 팀원에게 내가 이해한게맞는지

ppt를 만들어서 코드 한개 만들어서 설명하는 시간을 가져야겠다.

오늘 멘토님이 오셔서 팀원들의 질문을 잘 받아주셨다.

강사님을 매우 좋아하신다고한다.

팀원들과 만나서 정리해보자고 의견의 나왔다.

다들 동의하면서 시간되는날 만나서 팀원들과 서로 도움주고 해결하고 싶다.

> 앞으로 내가 해야할일은 복습을 꾸준히하는것이다.
>
> 오늘의 문구는
>
> " 천천히, 정확하고 부드럽게 하는 것이 결국 더 빠른 헤엄이다"