객체지향 프로그래밍
3장

클래스와 객체

조용주
ycho@smu.ac.kr

간단한 함수 만들어보기

- 클래스는 데이터와 그 데이터를 사용해서 특정 작업을 처리하는 함수가 함께 묶여 있는 사용자 정의 자료형(user-defined type)
  - 멤버 변수(member variable) – 데이터
  - 멤버 함수(member function) 또는 메소드(method) – 함수
- 객체(object) 또는 인스턴스(instance)
  - 클래스를 프로그램 실행 중에 사용할 수 있도록 생성한 것
- 이미 사용해본 객체와 메소드
  - System.out.println()과 System.out.printf()
  - Scanner객체의 next()나 nextInt() 등

간단한 함수 만들어보기

- 함수
  - 특정 작업을 처리하도록 만들어진 한 줄 이상의 코드 묶음 (code block)
  - 함수를 사용한다는 것은 자동차를 정비하는 사람, 은행원, 강사, 혹은 생활의 달인 같은 사람들에게 전문적인 일을 의뢰하는 것과 같이 전문적인 작업을 다른 사람에게 의뢰하는 것
  - println()함수는 주어진 내용을 화면에 출력
  - printf()함수는 주어진 서식에 맞춰 출력
  - next~()함수들은 각각 주어진 역할에 맞게 문자열, 정수, 실숫값 등을 입력 받음
  - 반복적인 작업에서 재사용 가능한 코드이기도 함
    - 예: 100번 정수를 입력 받음

간단한 함수 만들어보기

- 코드에서 함수를 사용하는 것을 호출한다고 함
  - 다른 사람에게 작업을 시키는 것과 비슷함
    - 사람에게 이름이 있듯이 함수에도 이름이 있음
      - 그 이름을 불러서(호출해서) 작업을 요청
    - 은행원에게 출금을 요청하면서 금액과 신분증을 주거나, 강사에게 특정 내용의 강의를 요청하는 것처럼, 함수에게 일을 시킬 때 필요한 정보를 전달할 수 있음
    - 은행 직원에게 출금을 요청했을 때, 돈을 받는 것처럼, 함수 내부 작업의 결과물을 받는 경우도 있음
  - 함수에 전달하는 정보
    - 입력, 인자, 또는 매개 변수
  - 돌려 받는 결과 값
    - 출력 또는 반환값(return value)

간단한 함수 만들어보기

- 함수 만드는 방법

```
반환_자료형 함수_이름(매개변수_리스트) {
    // 함수 코드
}
```

- 반환_자료형
  - return을 이용해서 반환
  - 없으면 void
- 매개변수_리스트
  - 변수 생성과 유사함
  - 한 개 이상이면 ','로 분리

간단한 함수 만들어보기

- 두 개 정수를 입력 받고, 합을 구해서 반환하는 함수를 구현

함수 요소	사람의 표현 방식	자바 표현 방식
이름	add	add
입력	정숫값, 정숫값	int 변수_이름, int 다른_변수_이름
결과	정숫값	int

- 함수로 구성

```
int add(int 변수_이름_1, int 변수_이름_2) {
    // 함수 코드가 여기에 들어감
}
```

간단한 함수 만들어보기

- 실제 코드

```
int add(int num1, int num2) {
    int res;
    res = num1 + num2;
    return res;
}
```

- JShell에서 실행시키기

간단한 함수 만들어보기

- 또 다른 예

```
// 입력은 있고, 결과 값이 없는 함수
// a와 b의 합을 구해서 화면에 출력함
// 반환값이 없으므로, return 키워드를 사용하지 않음
void addAndPrint(int a, int b) {
    int sum = a + b;
    System.out.println(sum);
}

// 입력이 없고, 결과 값이 없는 함수
// 매개 변수 리스트가 비어 있음.
void sayHello() {
    System.out.println("hello world");
}
```

프로그래밍 방법 2가지

- 객체 지향 프로그래밍(object-oriented programming)은?
  - 프로그래밍 하는 스타일
  - 기존 절차 중심 프로그래밍(procedural programming 또는 구조화 프로그래밍 structured programming)을 개선한 방법
- 프로그램 = 데이터 + 코드

절차 중심 프로그래밍

- 절차 중심 프로그래밍
  - 6-70년대 쓰였던 파스칼, C언어 등을 사용
  - 문제를 해결할 때 순서를 정해서 단계별로 작업하듯이, 코드를 주어진 절차 순서대로 실행
  - 복잡해지지 않고 재사용성을 높이기 위해 함수(function) 또는 프로시듀어(procedure)라는 작은 단위로 나누어서 작성
  - 절차와 데이터가 분리되어 있는 경우가 많고, 관리 어려움

절차 중심 프로그래밍

- 가솔린을 연료로 사용하는 내연기관 자동차를 절차 중심 프로그래밍 형태로 구성
  - 자동차의 내부 동작 구조
    - 엔진에 가솔린이 들어가고, 이를 연소시켜 발생하는 에너지가 자동차의 바퀴에 전달되어 돌아가도록 해서 차가 움직임 → "move"라는 이름으로 함수를 만듦
  - 차를 움직여야 할 때마다 move를 호출하면 됨
  - move 절차에는 가솔린이 필요함 → move 함수 내부에 포함시킬 수 없음
  - 입력으로 전달시키도록 함

절차 중심 프로그래밍

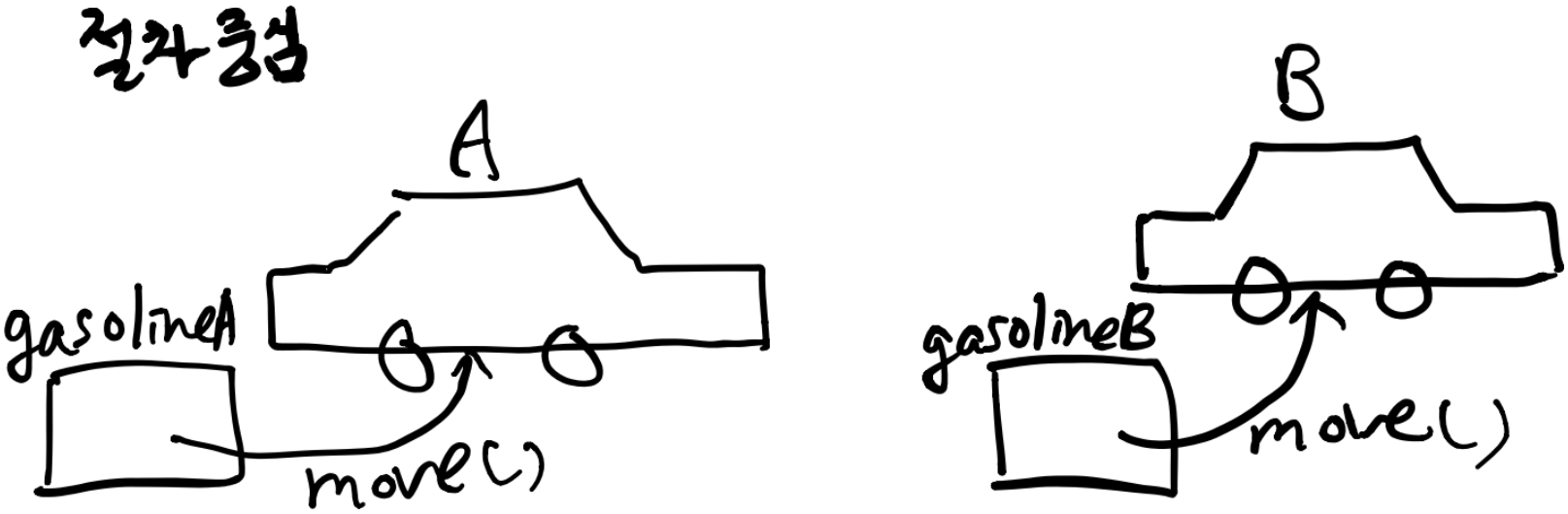
- 코드 예

```
void move(double gas) {
    // 여기에는 주어진 기름을 이용해서 엔진에서
    // 연소시키고 바퀴에 동력을 전달해서
    // 차를 움직이는 코드가 있다고 가정
    ...
}

double gasoline = 20.0; // 20리터의 가솔린이 있음
move(gasoline);         // 자동차를 움직임
```

절차 중심 프로그래밍

- 두 개 자동차에서 move() 함수를 사용한다면?

```
double gasolineA = 20.0; // A에 20리터 가솔린 있음
double gasolineB = 30.0; // B에 30리터 가솔린 있음

move(gasolineA);    // A자동차를 움직임
move(gasolineB);    // B자동차를 움직임
```



객체 지향 프로그래밍

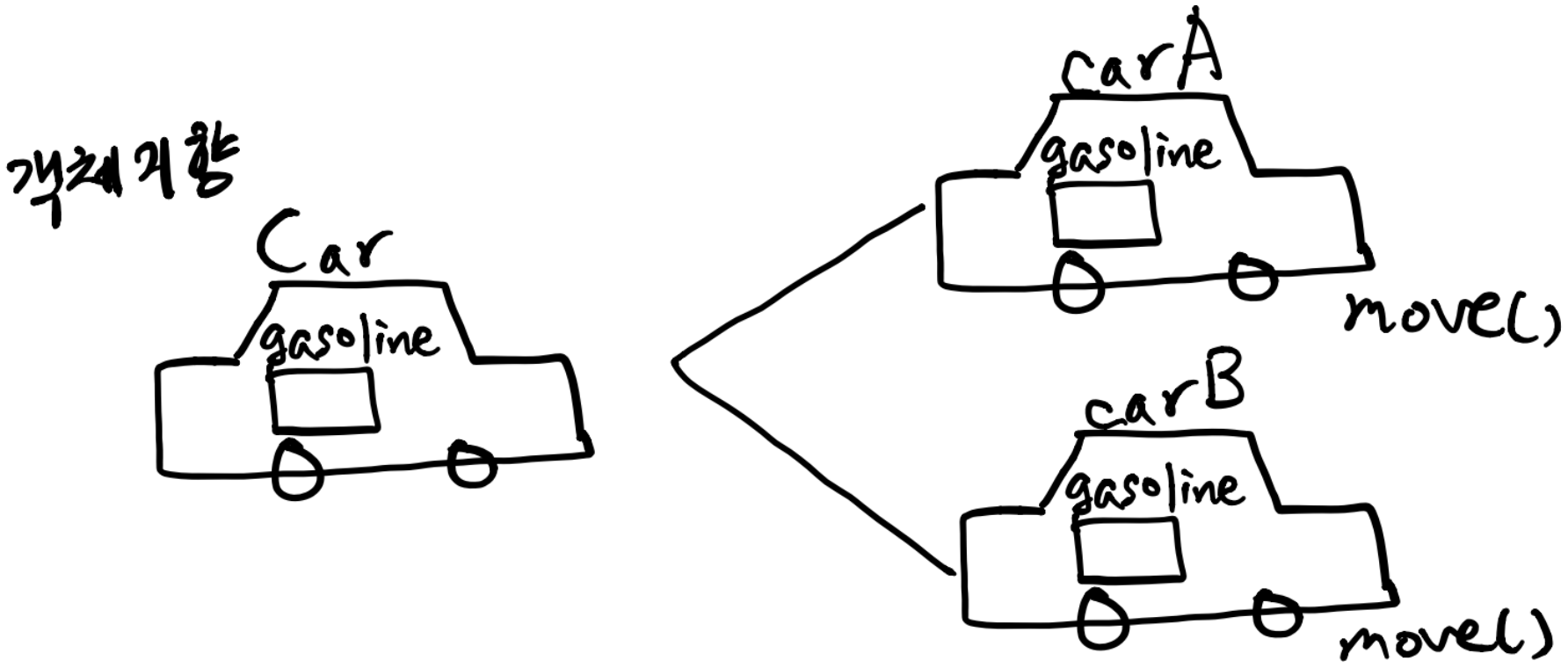
- 객체 지향 프로그래밍 방법에서는 데이터와 코드를 객체로 함께 구성해서 한 개의 자료형으로 취급할 수 있게 해줌
- 객체 지향 프로그래밍 방법은 새로 디자인된 언어에서 사용할 수 있으며, 주로 C++, Java, C#, Python, Ruby 같은 언어들처럼 80년대 이후에 만들어진 것들이 지원함
- 많은 종류의 객체 지향 프로그래밍 언어에서는 객체를 만들기 위해 클래스(class)라는 이름으로 데이터와 코드를 묶을 수 있는 기능을 제공함
- 자바스크립트(Javascript)는 클래스를 사용하지는 않지만, 또 다른 객체 생성 방법 제공

객체 지향 프로그래밍

- 객체 지향 코드로 다시 작성된 자동차와 가솔린

```
class Car {
    double gasoline;
    void move() {
        // 아래 코드는 주어진 기름을 이용해서
        // 엔진에서 연소시키고 바퀴에 동력을 전달해서
        // 차를 움직이는 코드가 있다고 가정
        ...
    }
}
Car carA = new Car();  // 객체 생성
Car carB = new Car();
carA.move();           // 메소드 호출
carB.move();
```

객체 지향 프로그래밍



객체 지향 프로그래밍

- 객체 지향 프로그래밍 방법은 데이터를 보호하고 코드를 재사용하기 쉽게 만든 방식
  - 절차 중심 프로그래밍 방법에서는 데이터와 함수가 떨어져 있음
    - 데이터는 프로그램의 다른 코드에서도 쉽게 읽고 수정할 수 있음
  - 객체 지향은 이러한 실수를 막고 코드의 재사용성을 높일 수 있음

절차 중심 프로그래밍	객체 지향 프로그래밍
C, Cobol, Fortran	C++, Java, C#, Python
절차를 순서대로 실행 코드와 데이터가 별도로 분리됨	객체를 만들어 실행 코드와 데이터가 묶여 있고 데이터를 보호

생각 바꾸기: 클래스도 명령문으로 구성됨

- 객체 지향 프로그래밍 기법
  - 프로그래밍 스타일 중 한 가지
  - 클래스를 사용하여 프로그램을 구현
- 프로그램은 명령문으로 이루어짐
  - 자세히 보면 2진수로 구성됨
- 객체 지향 프로그래밍 이전
  - 함수를 사용
  - 순서대로 명령문을 구성하고, 입출력이 관련 있는 경우 묶어서 함수로 구성
  - 주로 프로그램에서 요구되는 기능 위주로 동작 방법을 생각하고, 이들 기능에서 요구되는 데이터를 전달하는 형태로 프로그래밍

생각 바꾸기: 클래스도 명령문으로 구성됨

- 객체 지향 프로그래밍
  - 기능을 생각해서 함수로 만들고 필요한 데이터는 멤버 속성으로 구성
  - 함수와 속성을 함께 묶어서 객체로 구성
  - 객체는 함수와 속성을 함께 가지고 있음 → 함수에 데이터 즉 속성을 전달하지 않음
  - 객체에 있는 함수들은 함께 객체에 포함된 속성들을 자유롭게 접근하고 사용 가능

캡슐화(Encapsulation)

- 캡슐화(encapsulation)
  - 자세한 내부 사정을 드러내지 않고 외부에 보이는 부분만으로 충분히 사용할 수 있도록 만드는 작업
- 캡슐(capsule)
  - 작은 용기
  - 액체나 분말 형태의 약을 담아서 먹기 쉽게 만들어놓은 것
- 캡슐화의 예
  - 쓴 가루약을 담아 놓은 캡슐
  - 자동차

캡슐화(Encapsulation)

- 객체 지향 프로그래밍에서의 캡슐화
  - 데이터와 함수들을 클래스 내에 담고 외부 프로그래머가 사용할 수 있는 부분만 보이도록 하는 것
  - 캡슐화는 클래스 사용자로 하여금
    - 사용할 수 있는 부분만 확인해서 쉽게 사용
    - 내부 데이터를 보호
    - 내부 코드의 복잡함을 감춤
- 캡슐화의 목적
  - 클래스에서 제공하는 정해진 방법 외에는 외부에서 클래스 내부의 속성들을 변경시키지 못하게 막는 것
  - 클래스 내부에 구성된 함수의 구현 방법들을 외부에 노출시키지 않고 사용 방법만 제공하는 것
  - 접근 제어자를 이용해서 지정

왜 객체 지향 프로그래밍?

- 80년대 이후 객체 지향 프로그래밍은 개발 방법을 이끌어온 트렌드
  - PYPL이나 TIOBE 인덱스 등을 살펴보면 상위권에 분포되어 있는 언어들의 대부분이 객체 지향 언어임
  - C++나 파이썬 같은 일부 언어들은 객체 지향과 절차 중심 프로그래밍 방법을 모두 지원함
  - 자바나 C# 같은 객체 지향 언어들은 객체 지향 프로그래밍의 기본적인 내용을 모르면 프로그래밍 할 수 없음

왜 객체 지향 프로그래밍?

- 객체 지향은 왜 필요하며, 이와 같이 넓게 퍼지게 된 이유는 무엇일까?
  - 최초로 만들어진 객체 지향 프로그래밍 언어는 1960년대의 시뮬라 67
  - 객체 지향 기법은 개발되는 프로그램이 커지면서 빠르게 퍼짐
    - 바로 코딩 하기 보다는 구조를 설계해야 할 필요를 느끼게 됨

왜 객체 지향 프로그래밍?

- 객체 지향 기법은 기존 방식에 빠르게 만들면서, 품질이 뛰어난 결과물(프로그램)을 만들어낼 수 있는 것으로 알려져 있음
  - 추상화
    - 데이터와 실행 코드를 함께 묶어서 처리함으로써 독립적이고 재사용 가능한 코드 작성이 용이함
    - 클래스는 다른 곳에서 사용하기 쉽도록 추상화됨
  - 캡슐화
    - 어떻게 사용해야 하는지 공개된 부분만 알고 있으면 되며, 내부 코드나 데이터에 대해서 몰라도 됨
  - 재사용성
    - 상속, 다형성
    - 패턴

객체 지향 – 객체로 프로그래밍

- 객체 지향 프로그래밍 방법에서 코드는 객체로 구성
- 절차적/객체 지향적 프로그래밍

프로그래밍 방법	절차적 프로그램	객체 지향 프로그램
대표 언어	C, PHP	Java, C++
프로그램 구성 방법	**명령문을 함수로 구성**	**명령문을 클래스로 구성**

- 두 개의 숫자를 더해서 결과를 반환하는 프로그램을 C언어를 이용해서 절차적으로 구현

```
int add(int a, int b) {
    return a + b;
}
```

객체 지향 – 객체로 프로그래밍

- 객체 지향 프로그램은 간단한 기능이라도 클래스를 만들어야 함
  - 자바 언어로 구현

```
class Number {
    int add(int a, int b) {
        return a + b;
    }
}
```

클래스 형식

- 자바에서 객체를 만드는 방법은 클래스를 선언하는 것부터 시작됨
  - 클래스는 키워드 class로 시작
  - 데이터 – 멤버 변수
  - 절차 – 멤버 함수
- 형식

```
class 클래스명 {
      자료형 변수명;
      ...
      자료형 함수명(인자...) {

      }
}
```

객체와 클래스

- 객체 지향은 우리가 사는 세상에서 **실제로 일어나는 일을 흉내** 내어 프로그래밍 함
  - 주변에 있는 모든 것이 클래스가 될 수 있고 프로그래밍 될 수 있음
  - 명사(noun)로 표현될 수 있는 것들은 모두 객체로 표현할 수 있음
- **좋은 클래스**를 만드는 것이 중요함
  - 설계를 먼저 하는 것이 필요함
- **목적에 따라 클래스 내용이 달라질** 수 있음
  - 자동차나 커피 콩
  - **좋은 클래스란 필요한 데이터들만 포함할 수 있도록 응집성 있게** 만들어야 함

객체와 클래스

- 객체는 프로그램에서 사용할 수 있도록 클래스로부터 생성한 것
  - 앞에서 만들었던 Number 클래스는 어떤 데이터를 가지고 있고(멤버 변수 없음), 어떤 일을 할 수 있는지(add)에 대해서 보여줌
  - 실제 사용하려면 Number 객체를 생성해야 함
- 객체 생성 및 사용 예

```
jshell> class Number {
   ...>     int add(int a, int b) {
   ...>         return a + b;
   ...>     }
   ...> }
|  created class Number
```

객체와 클래스

```
jshell> Number num = new Number();
num ==> Number@5cb9f472

jshell> System.out.println(num.add(2, 3));
5
```

- Number 클래스는 객체를 만드는데 필요한 데이터와 함수들이 어떻게 생겼는지 알려주는 설명서 역할
- 객체 생성 방법

```
객체_변수 = new 클래스_이름()
```

- 객체 멤버 함수 사용 방법

```
객체_변수.멤버_함수_이름()
```

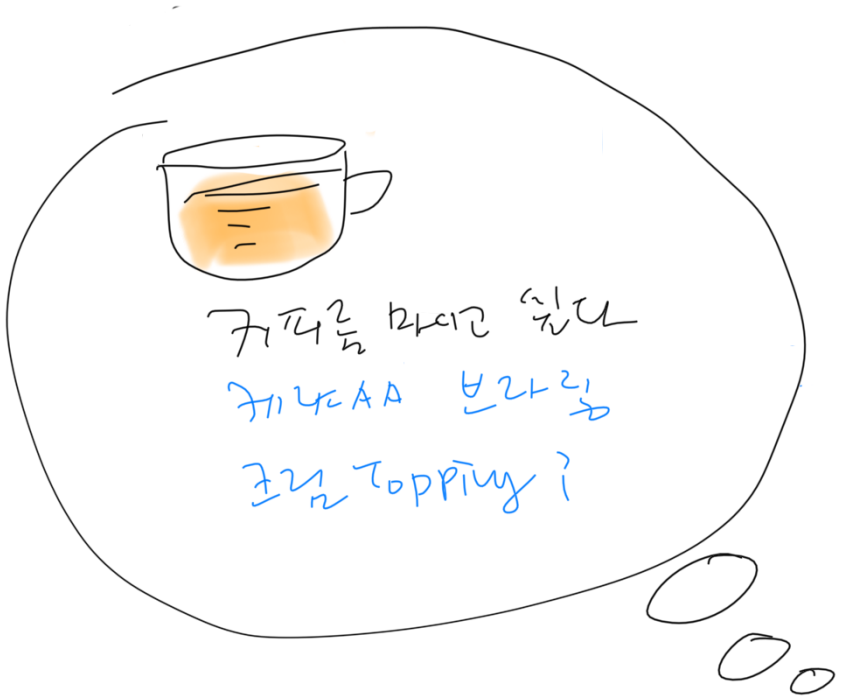
객체와 클래스

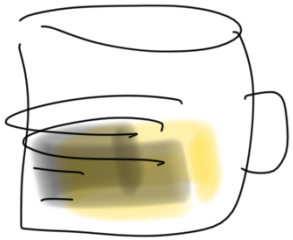
- 클래스는 객체가 어떤 속성(데이터)들을 가지고 있는지, 또 어떤 일들을 수행할 수 있는지 알려주는 일종의 **개념**이고 **상품 소개서**에 해당됨
- 객체(인스턴스)는 클래스의 **실제 상품**
  - 예: 자동차는 클래스, "내가 타는 차"는 객체, 커피는 클래스, "내 앞 탁자에 놓여 있는 커피"는 객체
- 인스턴스화(instantiation)
  - 클래스로부터 객체를 만들어내는 과정
  - 자동차를 에로 들면, 디자인, 모델명, 컬러, 엔진 배기량 등 속성들이 정해져 있고, 자동차가 무엇을 할 수 있는지 정해져 있음
  - 이러한 내용에 맞춰 자동차를 생산(인스턴스화)해서 소비자에게 인도(변수에 저장)

객체와 클래스

구분	클래스	객체
수(number)	Number	객체
자동차	버스, 택시, 화물차 어느 것이나 가능	내가 타는 자동차, 등록 번호를 가지고 있어 특정할 수 있는 자동차
커피	케냐AA, 브라질	내가 주문해서 지금 마시고 있는 커피
문서	이력서 서식, 논문 서식	내가 작성한 "mydocument.docx"

객체와 클래스





클래스는 객체의 속성과 함수를 정의

- 클래스는 객체의 속성과 함수를 선언
- 클래스의 속성
  - 자동차는 년식이라는 속성이 있음
    - 내 자동차에는 내가 구매한 년도라는 년식을 가지고 있음
  - 자동차에는 엔진 배기량 속성이 있음(1000, 2000, 3000cc)
    - 내가 구매한 자동차의 배기량은 정해져 있음
  - 자동차를 사면, 내 소유의 자동차를 증명할 수 있도록 등록해야 함 → 자동차 클래스에 절차로 포함시킬 수 있음
  - 자동차를 앞 또는 뒤로 이동시키는 작업, 멈추도록 하는 절차 등도 포함시킬 수 있음

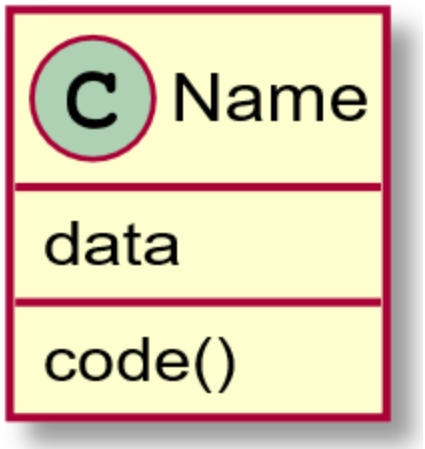
클래스는 객체의 속성과 함수를 정의

클래스	속성	함수
자동차	년식, 배기량	등록한다
커피	품종	추출한다

객체	속성	함수
내가 소유한 자동차	년식 = 2018 배기량 = 2000cc	등록한다
내가 들고 있는 커피	품종 = 케냐AA	추출한다

UML로 클래스 그려보기

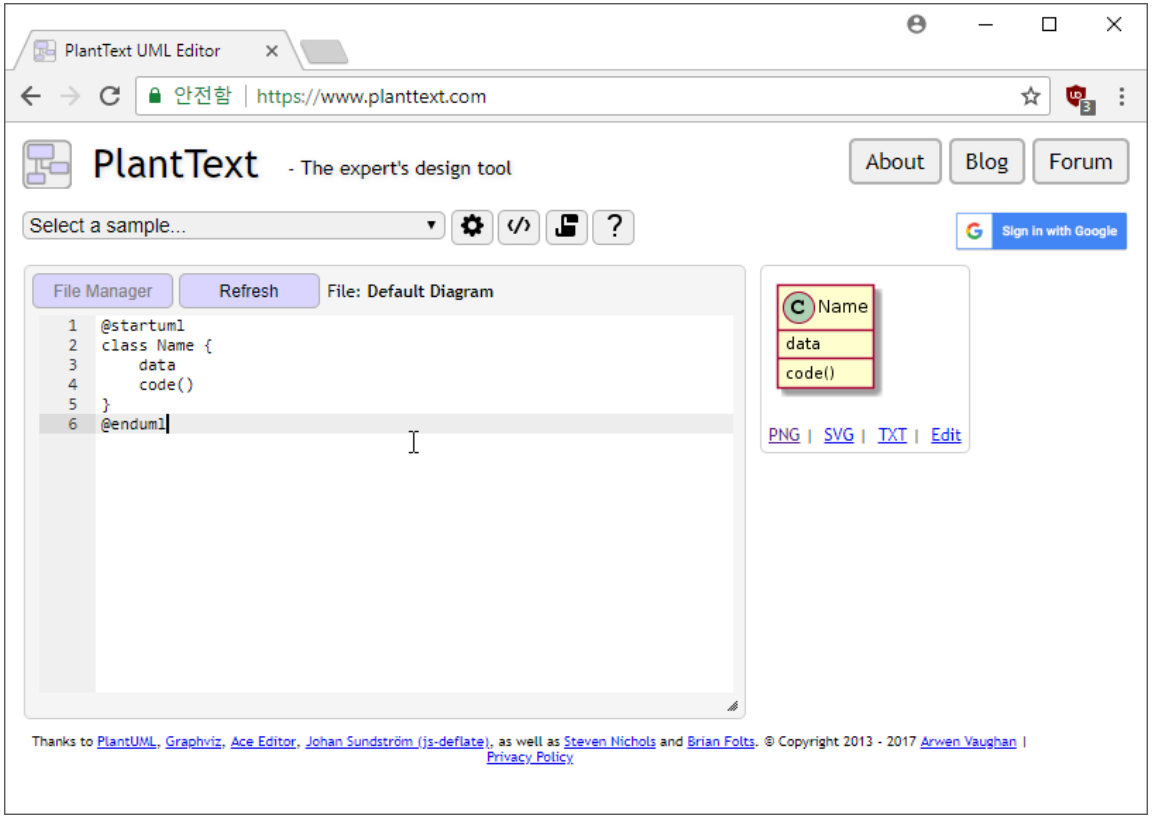
- 바로 클래스를 코드로 작성하는 것보다 그림으로 먼저 설계하는 것도 좋음
- UML(Unified Modeling Language)의 클래스 다이어그램(class diagram) 사용



  - UML에서 클래스는 3 부분으로 나누어짐
  - 첫 째 서랍에는 클래스 이름
  - 둘 째 칸에는 속성(변수와 값)
  - 세 째 서랍에는 함수가

UML로 클래스 그려보기

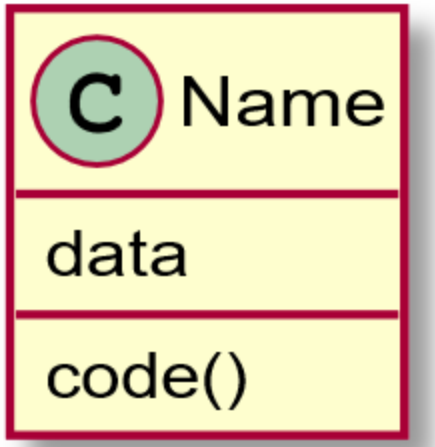
- 여기서는 PlantUML과 PlantUML을 웹에 구현한 planttext.com 사이트를 이용해서 그림을 그리려 함
  - http://www.planttext.com

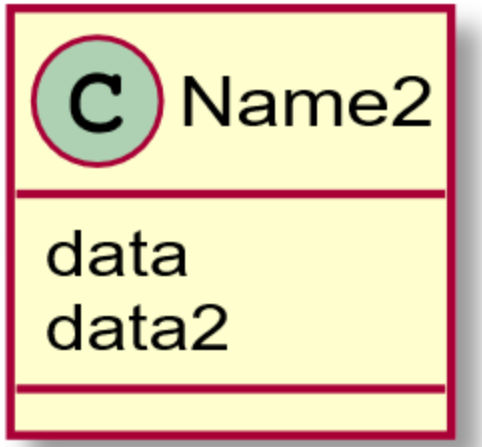


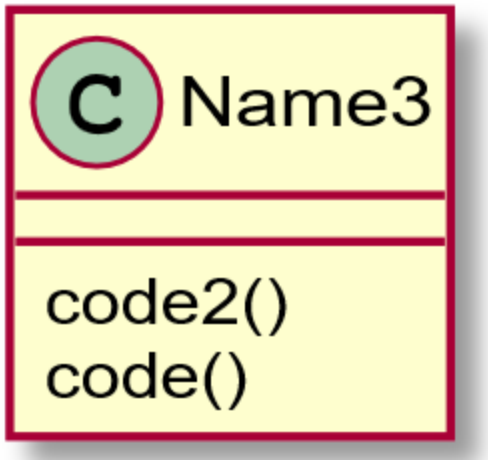

```
@startuml
class Name {
    data
    code()
}
@enduml
```

UML로 클래스 그려보기

```
@startuml
class Name {
    data
    code()
}
@enduml
@startuml
class Name2 {
    data
    data2
}
class Name3 {
    code2()
    code()
}
@enduml
```



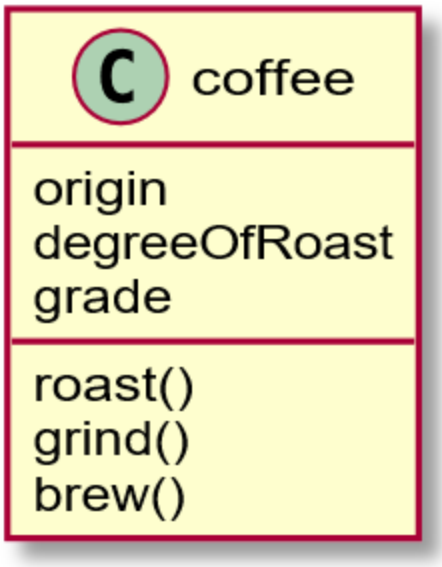




UML로 클래스 그려보기

- UML로 커피 클래스를 그려보기
  - 커피를 데이터와 코드로 분류
  - 데이터
    - 원산지(origin)
    - 로스트(degreeOfRoast) – 약배전, 중배전, 강배전
    - 등급(grade) – 커피 원두의 크기를 참조
  - 멤버 함수
    - 볶기(roast), 갈기(grind), 내리기(brew)

```
@startuml
class Coffee {
  origin
  degreeOfRoast
  grade
  roast()
  grind()
  brew()
}
@enduml
```



실습문제 1: Hello 프로그램으로 객체 지향, 절차 중심 방식 비교하기

- 문제
  - "hello"라는 인사말을 화면에 출력하는 프로그램을 함수와 클래스를 이용해서 구현
- 요구사항
  - 함수와 클래스를 만들고 화면에 출력하는데 필요한 부수적인 코드도 작성
- 해결
  - 인사를 출력하는 함수 구현
  - "함수 이름", "입력 내용", "반환값" 결정
    - 이름은 sayHello, 입력/반환값 없음

실습문제 1: Hello 프로그램으로 객체 지향, 절차 중심 방식 비교하기

- 함수 구현

```
jshell> void sayHello() {
   ...>
System.out.println("hello");
   ...> }
|  created method sayHello()
```

- 실행

```
jshell> sayHello();
hello
```

실습문제 1: Hello 프로그램으로 객체 지향, 절차 중심 방식 비교하기

- 클래스 구현
  - UML부터 작성

```
@startuml
class Hello {
   void sayHello()
}
@enduml
```

  - 클래스 작성

```
class Hello {
    void sayHello() {
        System.out.println("hello");
    }
}
```

실습문제 1: Hello 프로그램으로 객체 지향, 절차 중심 방식 비교하기

- JShell에서 실행

```
jshell> class Hello {
   ...>     void sayHello() {
   ...>
System.out.println("hello");
   ...>     }
   ...> }
|  created class Hello

jshell> Hello h = new Hello();

jshell> h.sayHello();
hello
```

실습문제 1: Hello 프로그램으로 객체 지향, 절차 중심 방식 비교하기

- 최종 코드

```
class Hello {
    void sayHello() {
        System.out.println("hello");
    }

    public static void main(String[] args) {
        Hello h = new Hello();
        h.sayHello();
    }
}
```

- 실행 결과

```
C:\DevTools\jdk-13.0.2\bin\java.exe "-javaa
hello
```

객체(object) 만들고 사용해보기

- 객체를 생성하는 것은 new 연산자를 사용

```
new 클래스_이름();
```

- Hello 클래스 객체를 생성

```
new Hello();
```

  - new
    - 객체를 생성하는 명령어
  - 클래스 이름을 따서 만든 함수
    - 생성자(constructor)

new 연산자와 힙(heap) 공간

- new 연산자는 힙(heap)이라는 메모리 공간에서 일부 영역을 할당 받은 뒤에 참조값(메모리 공간의 주소값)을 반환
- 힙은 프로그램 실행 중에 자유롭게 빌려서 사용하고 반납할 수 있는 메모리 공간
- 프로그래머가 필요한 크기 만큼 공간을 힙 관리자(heap manager)에게 요청
  - 공간이 있다면 공간을 대여하고 해당 공간의 시작 메모리 주소를 반환
  - 공간이 없다면 null을 반환
- 할당된 메모리 공간은 원할 때까지 또는 프로그램 종료 시까지 자유롭게 사용 가능

new 연산자와 힙(heap) 공간

- 자바에서는 더 이상 사용되지 않는 메모리 영역들을 찾아 힙 관리자에게 일괄적으로 반환하는 가비지 콜렉션(garbage collection) 기능을 제공
- 메모리 동적 할당 시스템
  - 프로그램 실행 중에 메모리 대여와 반납이 이루어지는 시스템

자바의 메모리 구조

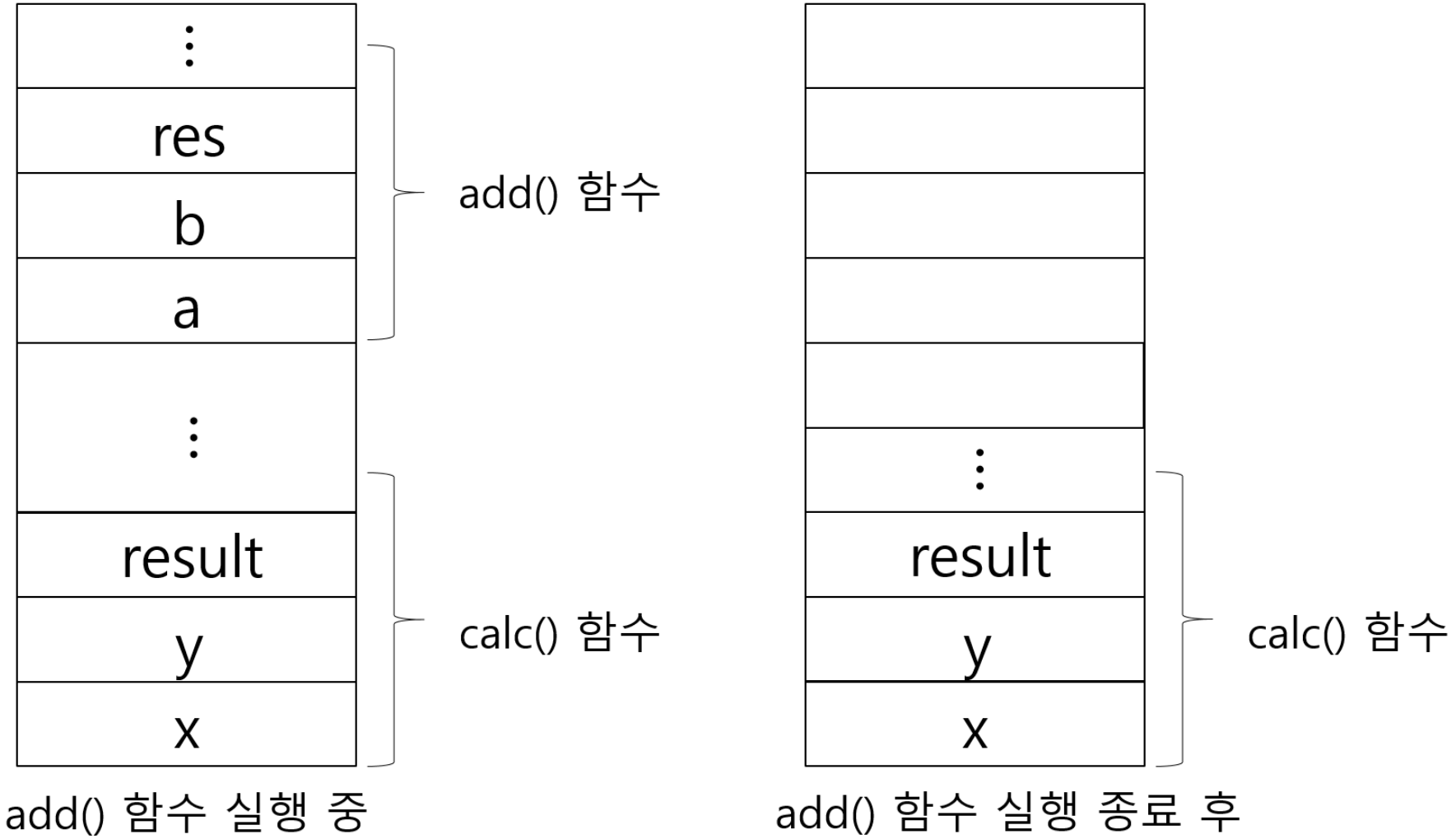
- 정적 메모리(static memory) 영역
  - 컴파일할 때 결정되고 실행 중에 변하지 않는 것들
  - 자바 코드, 클래스의 정적 멤버 변수나 멤버 함수, 상수 들이 저장되는 메모리 공간
- 스택(stack) 영역
  - 프로그램 실행 중에 증감하는 영역
  - 멤버 함수 내부에 생성된 지역 변수들과 매개 변수들이 저장됨
  - 함수가 호출되면 메모리가 할당되었다가 종료되면 사라짐

자바의 메모리 구조

```
int add(int a, int b) {
  int res = a + b;
  return res;
}

void calc() {
  int x = 3;
  int y = 5;
  int result = add(x, y);

  System.out.println(result);
}
```

자바의 메모리 구조

- 스택 영역 예시



자바의 메모리 구조

- 힙(heap) 영역
  - 메모리 공간을 필요에 따라 대여 받아서 사용하고 필요 없을 대 반환하는 시스템
  - 연속적인 메모리 공간을 대여함
  - 연속적인 공간이 없으면 null을 반환

```
void allocateObject() {
    Hello h = new Hello();
    ... // h 등을 사용하는 다른 코드
}
```

  - allocateObject() 함수는 코드이므로 정적 영역
  - h는 스택
  - Hello 객체는 힙

객체(object) 만들고 사용해보기

- 객체는 생성만으로는 사용할 수 없음

```
jshell> new Hello();
$9 ==> Hello@5f341870

jshell> new Hello();
$10 ==> Hello@47d0008

jshell> new Hello();
$11 ==> Hello@589838eb
```

- 객체를 사용하려면 구별할 수 있어야 함 → 이름을 붙여야 함

객체(object) 만들고 사용해보기

- 객체를 사용하기 위해 자바에서는 객체를 생성해서 변수에 저장하고, 변수의 이름을 이용해서 객체에 접근
  - 클래스가 자료형

```
// 첫 번째 방법
Hello h1;
h1 = new Hello();

// 두 번째 방법
Hello h2 = new Hello();
```

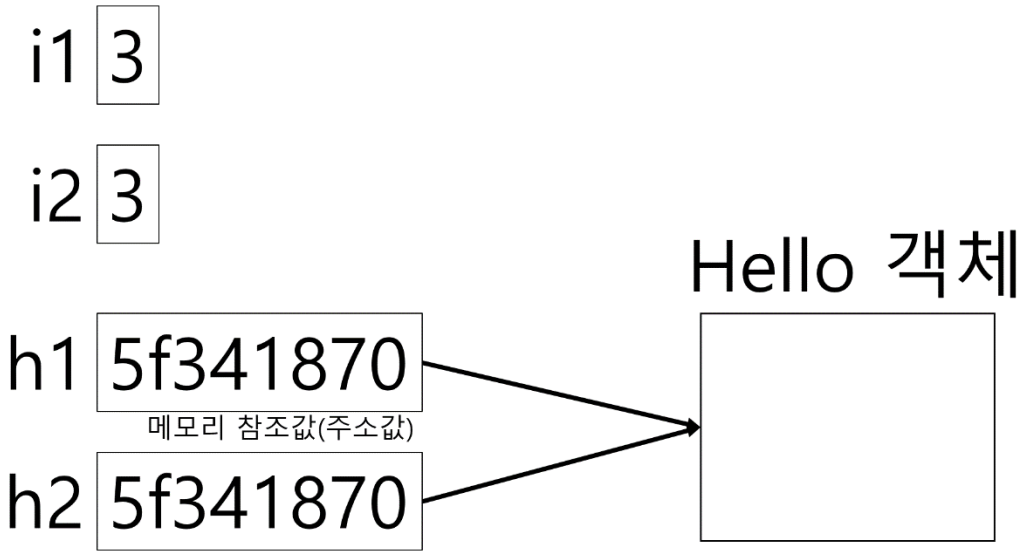
- 객체 생성 후에는 다른 변수에 저장하는 것도 가능

```
Hello h3 = h1;
```

기본형 변수는 값, 클래스 변수는 참조(값)

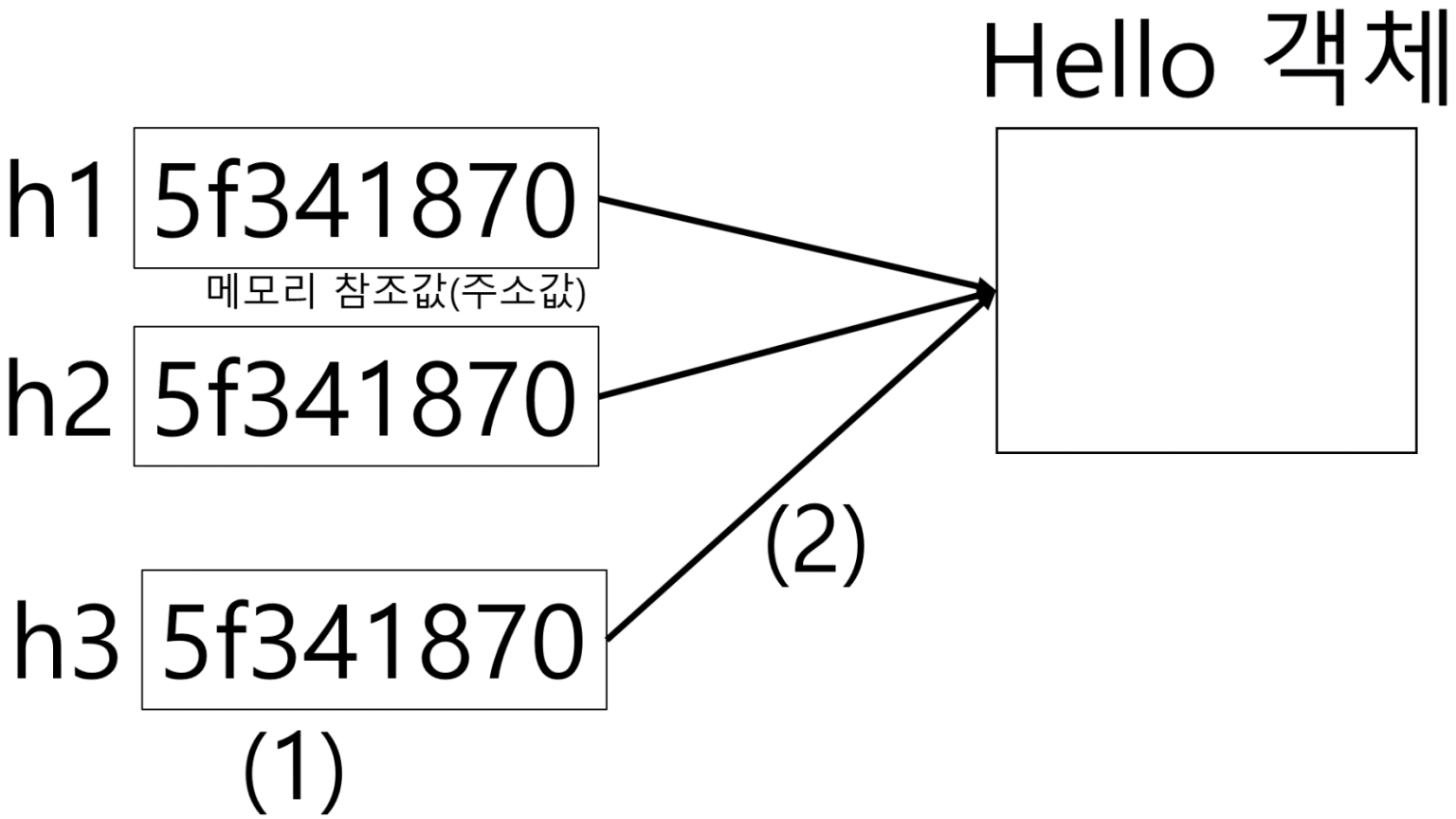
- 객체를 저장하는 변수는 참조값을 저장하는 참조형 (reference type)
  - 기본형 값을 저장하는 것과 다름
  - 객체를 변수에 저장하는 것은 해당 객체의 메모리 위치(주소)를 변수에 저장
- 기본형과 참조형의 차이

```
int i1 = 3;
int i2 = i1;

Hello h1 = new Hello();
Hello h2 = h1;
```



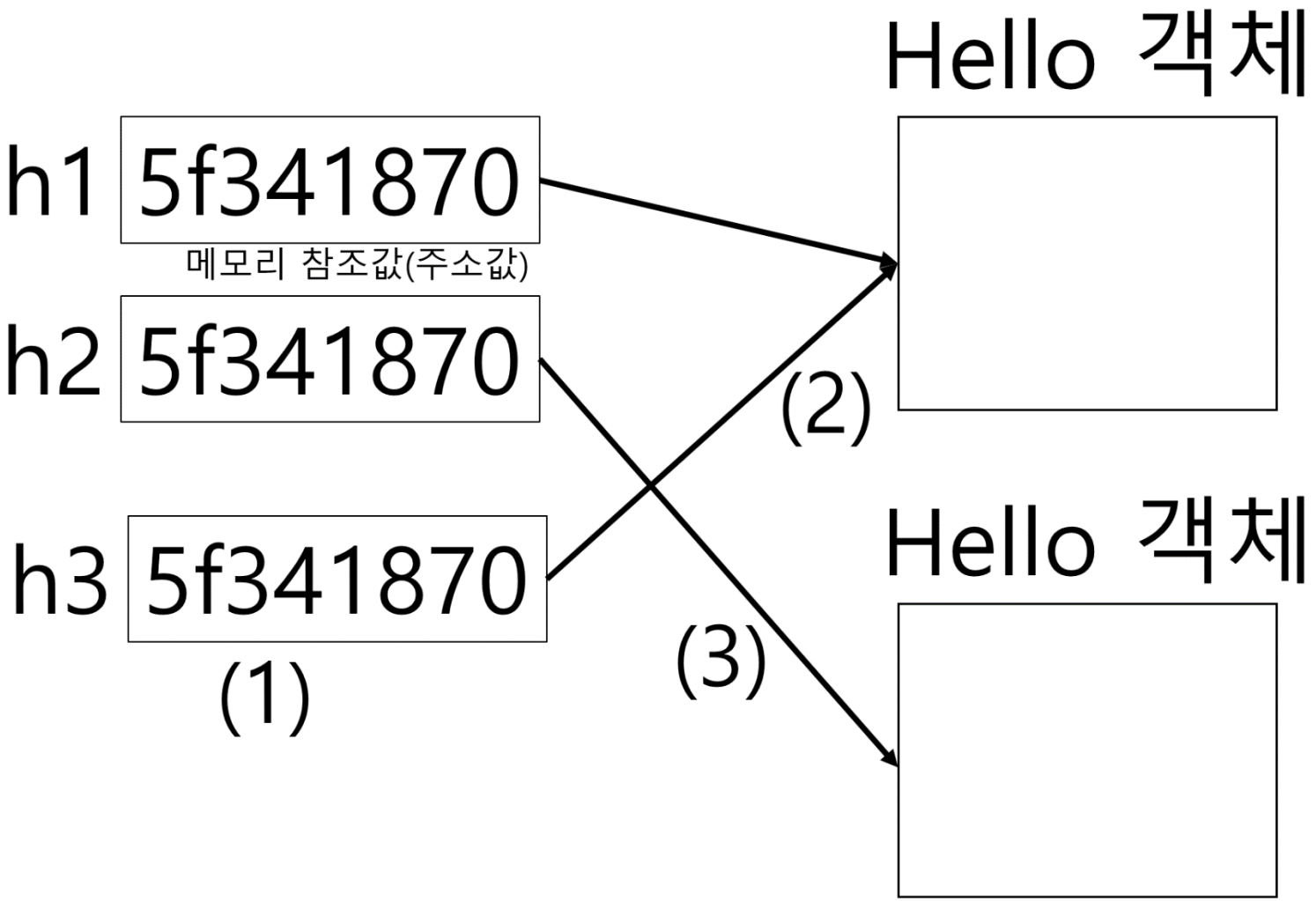
기본형 변수는 값, 클래스 변수는 참조(값)

```
Hello h3; // ----- (1)
h3 = h1;  // ----- (2)
```



기본형 변수는 값, 클래스 변수는 참조(값)

```
h2 = new Hello(); // ---- (3)
```



객체 사용하기

- 객체를 사용한다는 것은 객체에 포함되어 있는 변수에 접근하거나 메소드(함수)를 호출하는 것
  - 객체를 사용하려면 변수가 필요함
  - 객체 변수가 참조하는 객체의 멤버 변수나 함수에 접근하려면 점 연산자('.')를 이용함
    - 점 연산자를 기준으로 오른쪽에 있는 변수 또는 함수가 왼쪽 객체에 속해 있다는 의미를 가짐
  - 점 연산자를 이용해서 객체의 멤버 변수나 함수에 접근

```
객체변수이름.멤버_변수 _ 이름;
객체변수이름.멤버 _ 함수 _ 이름();
```

객체 사용하기

- 점 연산자를 사용해도, 객체의 모든 변수와 함수에 접근 가능한 것은 아님 → 클래스를 만드는 사람이 허용한 범위 내에서만 접근 가능
  - 객체의 접근성
  - 접근성을 허용하는 것이 접근 제어자(나중에 다루게 됨)
- 클래스의 멤버 변수에 직접 접근하는 것은 객체 지향 프로그래밍에서 별로 바람직하지 않음(캡슐화를 해침)
  - 당분간은 클래스의 멤버 함수를 이용하는 것만 살펴봄

객체 사용하기

- 다음 코드는 Hello 클래스 객체를 생성하고, 점 연산자를 이용해서 멤버 함수를 사용하는 것을 보임

```
jshell> Hello h1 = new Hello();
h1 ==> Hello@27efef64

jshell> h1.sayHello();
hello
```

  - Hello@27efef64에서 27efef64는 해시코드(나중에 다룸)

객체 사용하기

- 컴파일 해보기

```
// HelloTest.java
class Hello {
    void sayHello() {
        System.out.println("hello");
    }
}

public class HelloTest {
    public static void main(String[] args) {
        Hello h1 = new Hello();
        h1.sayHello();
    }
}
```

객체 사용하기

- Hello 객체 두 개를 만들어서 각각 sayHello() 메소드를 실행시켜 보기

```
jshell> Hello h1 = new Hello();

jshell> Hello h2 = new Hello();

jshell> h1.sayHello();
hello

jshell> h2.sayHello();
hello
```

객체 사용하기

```
// HelloTest2.java
class Hello {
    void sayHello() {
        System.out.println("hello");
    }
}

public class HelloTest2 {
    public static void main(String[] args) {
        Hello h1 = new Hello();
        h1.sayHello();
        Hello h2 = new Hello();
        h2.sayHello();
    }
}
```

문자열 상수(string constant)

- 문자열
  - 한 개 이상의 문자, 숫자(문자로 취급), 기호 등을 나열해 놓은 것
  - 자바에서 문자열을 코드에서 직접 표현하는 방법
    - 큰 따옴표 두 개 사이에 문자들을 넣음
- 문자열 상수
  - 코드에서 사용된 따옴표에 둘러싸인 문자열
  - 예

```
"hello"
"hello world!"
"login id:"
"passworld:"
"word12"
"pass@java.network"
```

문자열 상수(string constant)

- 문자열에서 큰 따옴표 기호를 표현하려면 이스케이프 시퀀스 사용

```
"He said, \"I love you\""
```

  - 이스케이프 시퀀스를 사용하지 않으면?

```
"He said, "I love you""
```

- String 클래스와 문자열 변수
  - 자바에서 문자열 값을 저장하려면 String 자료형을 사용
  - String은 클래스 자료형 (참조형)
  - 예시

```
String s = "hello";
s = "new world";
```

문자열 상수(string constant)

- 한 개 변수에 있던 값을 다른 변수에 저장 가능

```
jshell> String s =
"hello";

jshell> String t = s;

jshell>
System.out.println(s);
hello

jshell>
System.out.println(t);
hello
```

문자열 변수와 초기화

- String형 변수도 변수를 정의할 때 초기값을 지정하는 것이 좋음

```
String hello = "Hello";
System.out.println(hello);
```

- 초기화 시키지 않는다면?
  - JShell에서 초기화 시키지 않은 변수를 사용하는 예

```
jshell> String hello;
jshell> System.out.println(hello);
null

jshell> hello = "Hello";
jshell> System.out.println(hello);
Hello
```

문자열 변수와 초기화

- 초기화 시키지 않은 변수를 사용하는 코드를 컴파일?

```
// InitializeTest.java
public class InitializeTest {
    public static void main(String[] args) {
        String s;
        System.out.println(s); // s 출력
    }
}
```

  - 컴파일 오류

문자열 변수와 초기화

```
// InitializeTest2.java
public class InitializeTest2 {
    String s;

    void printString() {
        System.out.println(s); // s 출력
    }

    public static void main(String[] args) {
        InitializeTest2 s = new InitializeTest2();
        s.printString();
    }
}
```

```
C:\Code\java\03>java InitializeTest2
null
```

문자열과 덧셈 연산자(+)

- 사칙 연산자 중에서 덧셈(+) 연산자는 예외적으로 문자열에서도 사용할 수 있음
  - 피연산자로 제공되는 문자열 두 개를 순서대로 연결해서 새로운 문자열을 생성

```
jshell> String s1 = "Hello";

jshell> String s2 = " World";

jshell> System.out.println(s1 + s2);
Hello World

jshell> String newString = s1 + s2;

jshell> System.out.println(newString);
Hello World
```

문자열과 덧셈 연산자(+)

- '+'연산자의 결과는 또 다른 '+'연산자의 피 연산자로 사용될 수 있음

```
jshell> String newStr = s1 + s2 + "!";

jshell> System.out.println(newStr);
Hello World!
```

- 덧셈 연산자는 문자열과 숫자, 문자, 불린 값들과 더해지면 새로운 문자열을 생성
  - 두 개 피연산자 중 한 개 이상이 문자열
  - 나머지 한 개의 자료형이 정수, 실수, 불린형이라면 자바에서는 문자열형으로 변환하고 다른 문자열 피연산자와 붙여 새로운 문자열 생성

```
jshell> "abc" + 3;

jshell> "" + 3

jshell> 3.24 + ""

jshell> 3.1415 + "는 PI 값이다";

jshell> String s = 3.1415 + "는 PI 값이다";

jshell> System.out.println(s);

jshell> String newStr = s + '.';

jshell> System.out.println(newStr);

jshell> boolean b = 3 > 2;

jshell> String str = "3 > 2는 " + b + " 입니다";

jshell> System.out.println(str);
```

덧셈 연산자(+)의 결합 순서

- 덧셈 연산자의 결합 순서는 왼쪽 → 오른쪽
  - 예

```
2 + 3 + 5
```

- 문자열과 함께 사용되는 덧셈 연산자의 결합 순서도 동일

```
jshell> boolean b = 3 > 2;

jshell> String str = "3 > 2는 " + b + " 입니다";

jshell> System.out.println(str);

jshell> System.out.println(2 + 3 + " = 23일까?");

jshell> System.out.println("" + 2 + 3 + " = 23일까?");
```

String이 참조하는 문자열의 내용은 수정 불가

- String 변수가 참조하는 문자열의 내용은 바꿀 수 없음
  - 변숫값을 바꾼다는 것과 다른 의미
  - String형 변수에는 다른 문자열들이 저장될 수 있음
  - 하지만 String형 변수가 참조하는 문자열의 내용은 변경 불가

```
jshell> String str = "Hello";

jshell> System.out.print(str);

jshell> str = "World";

jshell> System.out.println(str);
```

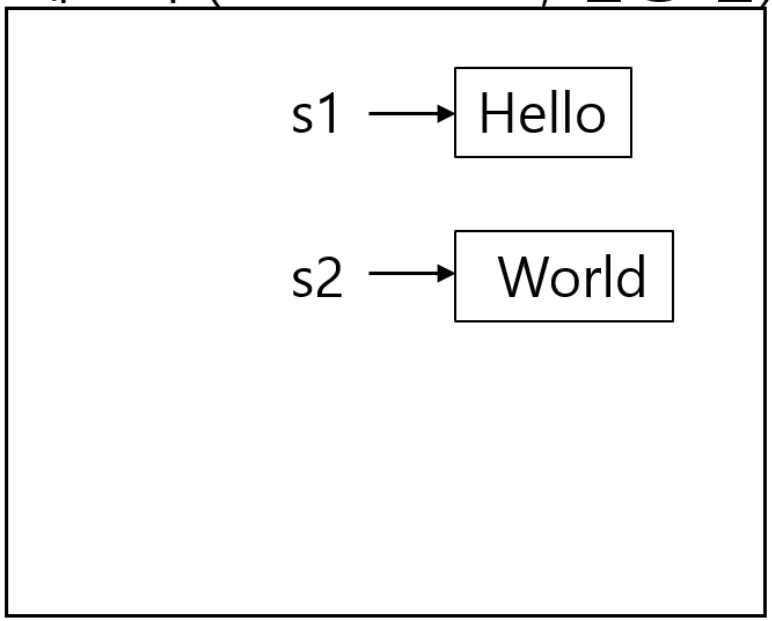
String이 참조하는 문자열의 내용은 수정 불가

- 문자열을 수정하고 싶으면 새로 만들어서 저장(참조)해야 함

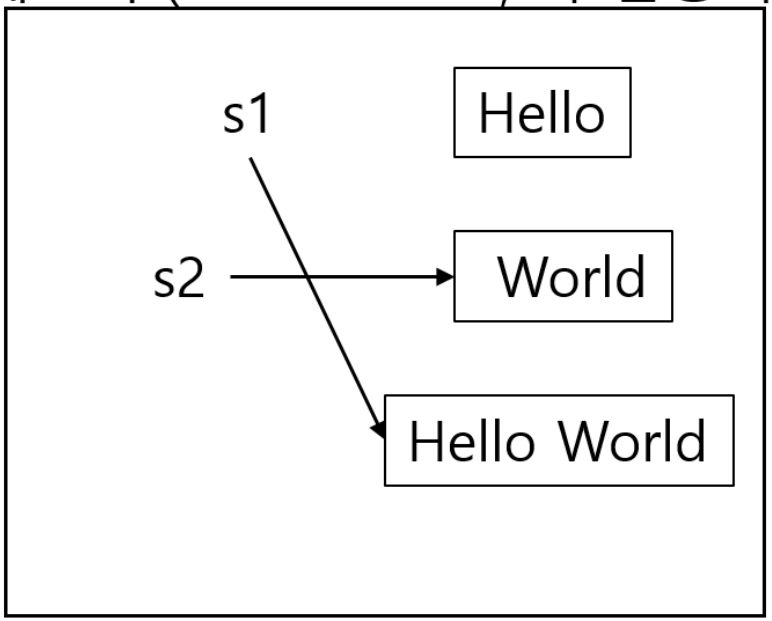
```
jshell> String s1 = "Hello";

jshell> String s2 = " World";

jshell> s1 = s1 + s2;
s1 ==> "Hello World"

jshell> System.out.println(s1);
Hello World
```

String이 참조하는 문자열의 내용은 수정 불가

메모리 (s1 = s1 + s2; 실행 전)



메모리 (s1 = s1 + s2; 가 실행 후)



String이 참조하는 문자열의 내용은 수정 불가

```
// StringTest2.java
public class StringTest2 {
    public static void main(String[] args) {
        String s1 = "Hello";
        String s2 = " World";
        s1 = s1 + s2;
        System.out.println(s1);
    }
}
```

```
C:\Code\java\03>java StringTest2
Hello World
```

String 클래스의 대표적 메소드

메소드	설명
int length()	문자열의 길이를 반환
char charAt(int index)	주어진 index위치(0부터 시작)의 글자 반환
int compareTo(String aStr)	- 객체의 문자열과 aStr을 비교. 글자 단위로 비교해서 같으면 0, aStr이 사전적으로 뒤에 나오면 양수, aStr이 사전적으로 앞에 있으면 음수가 반환됨 - 대문자와 소문자를 구별해서 비교함
int compareToIgnoreCase(String aStr)	- 객체의 문자열과 aStr을 비교. 글자 단위로 비교해서 같으면 0, aStr이 사전적으로 뒤에 나오면 양수, aStr이 사전적으로 앞에 있으면 음수가 반환됨 - 대문자와 소문자를 구별하지 않음

String 클래스의 대표적 메소드

메소드	설명
String concat(String str)	현재 객체의 문자열에 str을 붙인 새로운 문자열을 생성해서 반환

- 문자열 길이 출력

```
jshell> String s1 = "abc";

jshell> String s2 = "abcdef";

jshell> System.out.printf("\"abc\"의 길이: %d\n",
                          s1.length());

jshell> System.out.printf("\"abcdef\"의 길이: %d\n",
                          s2.length());
```

String 클래스의 대표적 메소드

- 특정 위치의 문자 반환

```
jshell> System.out.printf("\"abc\"의 두 번째 글자: %c\n", s1.charAt(1));

jshell> System.out.printf("\"abcdef\"의 세 번째 글자: %c\n", s2.charAt(2));
```

- 문자열 비교

```
jshell> System.out.printf("\"abc\"와 \"abcdef\"의 compareTo: %d\n", s1.compareTo(s2));

jshell> System.out.printf("\"abc\"와 \"ABC\"의 compareToIgnoreCase: %d\n",
s1.compareToIgnoreCase("ABC"));
```

String 클래스의 대표적 메소드

- 문자열 연결 ('+'연산자와 같은 일을 함)

```
jshell> System.out.printf("\"abc\"와 \"abcdef\"를
붙여서 새로운 문자열을 만들어냄\n");

jshell> String newStr = s1.concat(s2);

jshell> System.out.printf("newStr = %s\n", newStr);

jshell> System.out.printf("s1 = %s\n", s1);

jshell> System.out.printf("s2 = %s\n", s2);
```

수정할 수 있는 문자열 클래스

- 문자열의 내용을 직접 수정하고 싶으면(추가, 삭제, 변경 등) StringBuffer나 StringBuilder 클래스를 사용
  - StringBuffer나 StringBuilder는 완전히 같은 인터페이스 제공
  - StringBuffer는 다중쓰레드(multi-threaded) 프로그램을 작성할 때 동기화(synchronization)을 지원하고 StringBuilder는 그렇지 않음

수정할 수 있는 문자열 클래스

메소드	설명
StringBuilder append(String str)	객체 문자열에 str에 주어진 문자열을 추가(연결시킴)
char charAt(int index)	주어진 index위치(0부터 시작)의 글자를 반환
StringBuilder delete(int start, int end)	인덱스 번호 start부터 end - 1까지 삭제 (end에 있는 문자는 삭제 안됨)
char deleteCharAt(int index)	주어진 index위치(0부터 시작)의 글자를 삭제

수정할 수 있는 문자열 클래스

- StringBuilder는 반드시 new 연산자를 이용해서 객체를 생성한 후에 사용해야 함
  - String 사용하듯이 쓰면 오류 발생

```
jshell> StringBuilder sb = "Hello World";

|  Error:
|  incompatible types: java.lang.String cannot be
converted to java.lang.StringBuilder
|  StringBuilder sb = "Hello World";
|                     ^-----------^
```

수정할 수 있는 문자열 클래스

- new 연산자를 이용해서 다시 생성

```
jshell> StringBuilder sb = new StringBuilder("Hello");
sb ==> Hello

jshell> sb.append(" World");
$63 ==> Hello World

jshell> System.out.println(sb);
Hello World
```

문자열로부터 정수, 실숫값 입력 받기

- 기본형의 포장 클래스에는 문자열에서 숫자값을 추출하는 함수들을 제공
  - 문자열이 숫자로만 구성되어야 함
- Scanner 클래스를 이용해서 입력 버퍼를 통해서 입력 받는 것처럼 문자열에서 입력 받는 것이 가능함
  - 입력 버퍼의 역할을 문자열이 함
  - Scanner 객체 생성시 문자열을 소스로 전달

```
import java.util.Scanner;

String str = "1 2";

Scanner sc = new Scanner(str);
```

문자열로부터 정수, 실숫값 입력 받기

- 주어진 문자열 "1 2"에서 문자열과 정숫값을 추출하는 코드

```
String s = sc.next();
int n = sc.nextInt();

System.out.println(s);
System.out.println(n);
```