

# 생성자

# 생성자

---

## ❖ 생성자

- new 연산자에 의해 호출되어 객체의 초기화 담당

`new 클래스명();`

- 필드의 값 설정
- 메소드 호출해 객체를 사용할 수 있도록 준비하는 역할 수행

# 생성자

## ❖ 기본 생성자(Default Constructor)

- 모든 클래스는 생성자가 반드시 존재하며 하나 이상 가질 수 있음
- 생성자 선언을 생략하면 컴파일러는 다음과 같은 기본 생성자 추가

```
[public] 클래스() { }
```

소스 파일(Car.java)

```
public class Car {  
  
}
```



바이트 코드 파일(Car.class)

```
public class Car {  
    public Car() { } //자동 추가  
}          기본 생성자
```

```
Car myCar = new Car();
```

기본 생성자

# 생성자

---

## ❖ 생성자 선언

- 디폴트 생성자 대신 개발자가 직접 선언

```
클래스( 매개변수선언, ... ) {  
    //객체의 초기화 코드  
}
```

} 생성자 블록

# 생성자

---

## ❖ 생성자 선언

- 개발자 선언한 생성자 존재 시 컴파일러는 기본 생성자 추가하지 않음

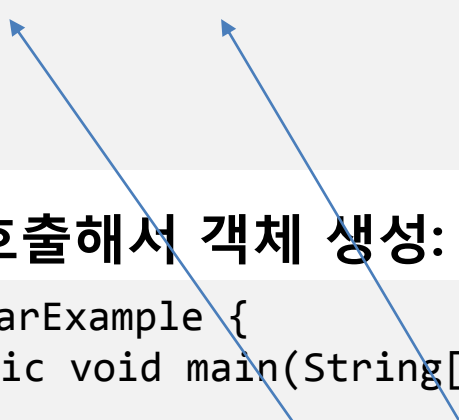
```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

```
Car myCar = new Car("그랜저", "검정", 300);
```

# 생성자

## ❖ 생성자 선언: Car.java

```
public class Car {  
    // 생성자  
    Car(String color, int cc) {  
    }  
}
```



## ❖ 생성자를 호출해서 객체 생성: CarExample.java

```
public class CarExample {  
    public static void main(String[] args) {  
  
        Car myCar = new Car("검정", 3000);  
  
        //Car myCar = new Car(); (x)  
    }  
}
```

기본 생성자가 정의되지 않았으므로  
기본 생성자를 호출할 수 없음

# 생성자

---

## ❖ 필드 초기화

- 초기값 없이 선언된 필드는 객체가 생성될 때 기본값으로 자동 설정

```
public class Korean {  
    String nation = "대한민국";  
    String name;  
    String ssn;  
}
```

```
Korean k1 = new Korean();  
Korean k2 = new Korean();
```

# 생성자

## ❖ 다른 값으로 필드 초기화하는 방법

- 필드 선언할 때 초기값 설정
- 생성자의 매개값으로 초기값 설정
- 매개 변수와 필드명 같은 경우 `this` 사용

```
public class Korean {  
  
    //필드  
    String nation = "대한민국";  
    String name;  
    String ssn;  
  
    // 생성자  
    public Korean(String n, String s) {  
        name = n;  
        ssn = s;  
    }  
}
```

Korean k1 = new Korean("박자바", "011225-123456");  
Korean k2 = new Korean("김자바", "930525-0654321");



# 생성자

## ❖ 생성자에서 필드 초기화: Korean.java

```
public class Korean {  
    // 필드  
    String nation = "대한민국";  
    String name;  
    String ssn;  
  
    public Korean(String name, String ssn) {  
        this.name = name;  
        this.ssn = ssn;  
    }  
}
```

```
public Korean(String name, String ssn) {  
    this.name = name;  
        필드      매개 변수  
    this.ssn = ssn;  
        필드      매개 변수  
}
```

# 생성자

---

## ❖ 객체 생성 후 필드값 출력: KoreanExample.java

```
public class KoreanExample {  
    public static void main(String[] args) {  
  
        Korean k1 = new Korean("박자바", "011225-1234567");  
        System.out.println("k1.name : " + k1.name);  
        System.out.println("k1.ssn : " + k1.ssn);  
  
        Korean k2 = new Korean("김자바", "930525-0654321");  
        System.out.println("k2.name : " + k2.name);  
        System.out.println("k2.ssn : " + k2.ssn);  
    }  
}
```

# 생성자

## ❖ 생성자를 다양화해야 하는 이유

- 객체 생성할 때 외부 값으로 객체를 초기화할 필요
- 외부 값이 어떤 타입으로 몇 개가 제공될 지 모름
  - 여러 버전의 생성자가 준비

```
public class 클래스 {  
    클래스 ( [타입 매개변수, ...] ) {  
        ...  
    }  
  
    클래스 ( [타입 매개변수, ...] ) {  
        ...  
    }  
}
```

### [생성자의 오버로딩]

매개변수의 타입, 개수, 순서가 다르게 선언

## 생성자

---

### ❖ 생성자를 다양화해야 하는 이유

```
public class Car {  
    Car() { ... }  
    Car(String model) { ... }  
    Car(String model, String color) { ... }  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

```
Car car1 = new Car();  
Car car2 = new Car("그랜저");  
Car car3 = new Car("그랜저", "흰색");  
Car car4 = new Car("그랜저", "흰색", 300);
```

```
Car(String model, String color) { ... }  
Car(String color, String model) { ... } //오버로딩이 아님
```

# 생성자

## ❖ 생성자의 오버로딩: Car.java

```
public class Car {  
    // 필드  
    String company = "현대자동차";  
    String model;  
    String color;  
    int maxSpeed;  
  
    // 생성자  
    Car() {  
    }  
  
    Car(String model) {  
        this.model = model;  
    }  
  
    Car(String model, String color) {  
        this.model = model;  
        this.color = color;  
    }  
}
```

```
    Car(String model, String color,  
          int maxSpeed) {  
        this.model = model;  
        this.color = color;  
        this.maxSpeed = maxSpeed;  
    }  
}
```

# 생성자

---

## ❖ 객체 생성 후 필드값 출력: CarExample.java

```
public class CarExample {  
  
    public static void main(String[] args) {  
  
        Car car1 = new Car();  
        System.out.println("car1.company : " + car1.company);  
        System.out.println();  
  
        Car car2 = new Car("자가용");  
        System.out.println("car2.company : " + car2.company);  
        System.out.println("car2.model : " + car2.model);  
        System.out.println();  
  
        Car car3 = new Car("자가용", "빨강");  
        System.out.println("car3.company : " + car3.company);  
        System.out.println("car3.model : " + car3.model);  
        System.out.println("car3.color : " + car3.color);  
        System.out.println();  
    }  
}
```

## 생성자

---

### ❖ 객체 생성 후 필드값 출력: CarExample.java

```
Car car4 = new Car("택시", "검정", 200);
System.out.println("car4.company : " + car4.company);
System.out.println("car4.model : " + car4.model);
System.out.println("car4.color : " + car4.color);
System.out.println("car4.maxSpeed : " + car4.maxSpeed);
    }
}
```

# 생성자

---

## ❖ 다른 생성자 호출( `this()` )

- 생성자 오버로딩되면 생성자 간의 중복된 코드 발생
- 초기화 내용이 비슷한 생성자들에서 이러한 현상을 많이 볼 수 있음
  - 초기화 내용을 한 생성자에 몰아 작성
  - 다른 생성자는 초기화 내용을 작성한 생성자를 `this(...)`로 호출



# 생성자

## ❖ 다른 생성자 호출( this() )

```
Car(String model) {  
    this.model = model;  
    this.color = "은색";  
    this.maxSpeed = 250;  
}  
  
Car(String model, String color) {  
    this.model = model;  
    this.color = color;  
    this.maxSpeed = 250;  
}  
  
Car(String model, String color, int maxSpeed) {  
    this.model = model;  
    this.color = color;  
    this.maxSpeed = maxSpeed;  
}
```

} 중복 코드

} 중복 코드

} 중복 코드

# 생성자

## ❖ 다른 생성자를 호출해서 중복 코드 줄이기: Car.java

```
public class Car {  
    String company = "현대자동차";  
    String model;  
    String color;  
    int maxSpeed;  
  
    Car() { }  
  
    Car(String model) {  
        this(model, null, 0);  
    }  
  
    Car(String model, String color) {  
        this(model, color, 0);  
    }  
  
    Car(String model, String color, int maxSpeed) {  
        this.model = model;  
        this.color = color;  
        this.maxSpeed = maxSpeed;  
    }  
}
```

