02

객체와 생성자

Dart

클래스와 객체

클래스 선언과 생성

- class라는 예약어로 선언
- 객체를 생성할 때는 new 연산자를 이용해도 되고 생략해도 됩니다.

• 클래스 선언

```
class User {
  String name = 'kkang';
  int age = 10;

  void sayHello() {
    print('Hello $name, age: $age');
  }
}
```

• 객체 생성

```
User user1 = new User();
```

클래스와 객체

객체 멤버와 클래스 멤버

- 객체 멤버와 클래스 멤버(혹은 정적 멤버)로 구분
- 객체 멤버는 생성된 객체를 이용해서 접근

• 객체 멤버 이용

```
User user1 = User();
user1.sayHello();
user1.name = 'kim';
user1.age = 20;
```

- 클래스 멤버는 static 예약어로 선언한 멤버

• 클래스 멤버 선언

```
class MyClass {
  String data1 = 'hello';
  static String data2 = 'hello';

  myFun1() {
    print('myFun1 call....');
  }
  static myFun2() {
    print('myFun2 call....');
  }
}
```

클래스와 객체

객체 멤버와 클래스 멤버

- 객체 맴버는 객체를 생성하고 그 이름으로 접근

• 객체 멤버 이용

```
MyClass.data1 = 'world';   // 오류
MyClass obj = MyClass();
obj.data1 = 'world';       // 성공
```

- static으로 선언한 클래스 멤버는 클래스 이름으로 접근

• 클래스 멤버 이용

```
MyClass.data2 = 'world';   // 성공
MyClass obj = MyClass();
obj.data2 = 'world';       // 오류
```

생성자와 멤버 초기화

생성자 선언

- 생성자constructor는 클래스에 선언되어 객체를 생성할 때 호출
- 개발자가 만들지 않으면 컴파일러가 자동으로 클래스와 같은 이름으로 기본 생성자를 만들어 줍니다.

• 클래스 선언

```
class User {
}
```

• 기본 생성자를 추가한 예

```
class User {
  User() { }
}
```

생성자와 멤버 초기화

멤버 초기화

- 생성자는 보통 멤버를 초기화하는 용도로 사용

• 멤버 초기화 생성자

```
class User {
  late String name;
  late int age;
  User(String name, int age) {
    this.name = name;
    this.age = age;

  }
  sayHello() {
    print('name : $name, age : $age');
  }
}
```

생성자와 멤버 초기화

멤버 초기화

- 매개변수로 멤버를 초기화하는 생성자는 this 예약어를 이용하면 간단하게 작성

• 멤버 초기화 생성자 단순화

```
class User {
  late String name;
  late int age;
  User(this.name, this.age);

  sayHello() {
    print('name : $name, age : $age');
  }
}
```

생성자와 멤버 초기화

초기화 목록

- 초기화 목록은 생성자 선언부를 콜론(:)으로 구분하여 오른쪽에 작성
- 리스트에서 특정 항목을 선택하거나 함수 호출로 멤버를 초기화할 때 사용
- 다른 생성자를 this(), super() 등으로 호출하는 구문을 작성

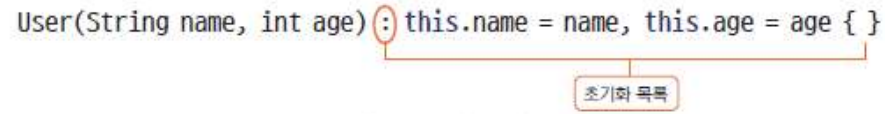

그림 6-1 생성자의 초기화 목록

• 리스트의 데이터로 초기화

```
class MyClass {
  late int data1;
  late int data2;

  MyClass(List<int> args)
      : this.data1 = args[0],
        this.data2 = args[1] { }
}
```

생성자와 멤버 초기화

초기화 목록

• 클래스 멤버 함수의 반환값으로 초기화

```
class MyClass {
  late int data1;
  late int data2;

  MyClass(int arg1, int arg2)
      : this.data1 = calFun(arg1),
        this.data2 = calFun(arg2) { }

  static int calFun(int arg) {
    return arg * 10;
  }

  printData() {
    print('$data1, $data2');
  }
}
```

명명된 생성자

- 명명된 생성자는 이름이 있는 생성자라는 의미
- 명명된 생성자는 한 클래스에 이름을 다르게 해서 여러 생성자를 정의하는 기법

그림 6-2 명명된 생성자 선언

• 명명된 생성자 선언

```
class MyClass {
  MyClass() { }
  MyClass.first() { }
  MyClass.second() { }
}
```

• 명명된 생성자로 객체 생성

```
var obj1 = MyClass();
var obj2 = MyClass.first();
var obj3 = MyClass.second();
```

명명된 생성자

this()로 다른 생성자 호출하기

- 한 클래스에 생성자를 여러 개 선언하면 생성자에서 다른 생성자를 호출

• this() 잘못된 호출 예

```
class MyClass {
  MyClass(int data1, int data2) {
    print('MyClass() call....');
  }
  MyClass.first(int arg) {
    this(arg, 0);    // 오류
  }
}
```

- 초기화 목록에 this() 호출문을 작성하면 생성자 본문을 작성할 수 없습니다.

• this() 잘못된 호출 예

```
class MyClass {
  MyClass(int data1, int data2) {
    print('MyClass() call....');
  }
  MyClass.first(int arg) : this(arg, 0) { }   // 오류
}
```

• 기본 생성자 중첩 호출

```
class MyClass {
  MyClass(int data1, int data2) {
    print('MyClass() call....');
  }
  MyClass.first(int arg) : this(arg, 0);   // 성공
}
```

명명된 생성자

this()로 다른 생성자 호출하기

- this() 호출문 외에 다른 구문을 사용할 수 없습니다.

• this() 잘못된 호출 예

```
MyClass.first(int arg) : this(arg, 0), this.data1=arg1;   // 오류
```

- 클래스에 작성된 명명된 생성자도 this()로 호출할 수 있습니다.

• 명명된 생성자 중첩 호출

```
class MyClass {
  late int data1;
  late int data2;
  MyClass(this.data1, this.data2);
  MyClass.first(int arg) : this(arg, 0);   // 기본 생성자(MyClass) 호출
  MyClass.second() : this.first(0);        // 명명된 생성자(MyClass.first) 호출
}
```

팩토리 생성자

- 팩토리 생성자factory constructor는 factory 예약어로 선언
- 생성자 호출만으로 객체가 생성되지는 않습니다.
- 팩토리 생성자에서 적절한 객체를 반환

• 팩토리 생성자 잘못된 선언 예

```
class MyClass {
  factory MyClass() {   // 오류
  }
}
```

팩토리 생성자

- 클래스 이름이 MyClass이므로 팩토리 생성자의 반환 타입은 MyClass로 고정
- MyClass는 널 불허로 선언했으므로 null을 반환할 수 없어서 오류가 발생

• 잘못된 객체 반환 예

```
class MyClass {
  factory MyClass() {
    return null;   // 오류
  }
}
```

• 팩토리 생성자 올바른 예

```
class MyClass {
  MyClass._instance();
  factory MyClass() {
    return MyClass._instance();
  }
}
main() {
  var obj = MyClass();
}
```

상수 생성자

const로 생성자 선언

- 상수 생성자constant constructor는 다음처럼 const 예약어로 선언하며 본문을 가질 수 없습니다.

• 상수 생성자 선언

```
class MyClass {
  const MyClass();
}
```

상수 생성자가 선언된 클래스의 모든 멤버 변수는 final로 선언

• 상수 생성자 잘못 선언한 예

```
class MyClass {
  int data1;
  const MyClass();   // 오류
}
```

상수 생성자

const로 생성자 선언

- 상수 생성자도 객체를 생성할 수 있으며 필요하다면 여러 개의 객체를 생성할 수도 있습니다.

• 상수 생성자의 객체 생성

```
class MyClass {
  final int data1;
  const MyClass(this.data1);
}

main() {
  var obj1 = MyClass(10);
  var obj2 = MyClass(20);
  print('obj1.data : ${obj1.data1}, obj2.data : ${obj2.data1}');
}
```

▶ 실행 결과

```
obj1.data : 10, obj2.data : 20
```

상수 생성자

const로 객체 생성

- const로 객체를 생성하려면 생성자 또한 const로 선언해야 합니다.

• 상수 객체 생성 오류

```
class MyClass { }

main() {
  var obj1 = const MyClass();   // 오류
}
```

• 상수 객체 생성

```
class MyClass {
  final int data1;
  const MyClass(this.data1);
}

main() {
  var obj1 = const MyClass(10);
}
```

상수 생성자

const로 객체 생성

- 상수 객체로 선언하면서 생성자에 전달한 값(초깃값)이 똑같으면 객체를 다시 생성하지 않고 이전 값으로 생성한 객체를 그대로 사용

• 같은 값으로 상수 객체 선언

```
var obj1 = const MyClass(10);
var obj2 = const MyClass(10);
print('obj1 == obj2 : ${obj1 == obj2}');  // true
```

- 객체를 생성할 때 전달하는 초깃값이 다르면 서로 다른 객체가 생성

• 다른 값으로 상수 객체 선언

```
var obj1 = const MyClass(10);
var obj2 = const MyClass(20);
print('obj1 == obj2 : ${obj1 == obj2}');   // false
```

• 같은 값으로 상수 객체와 일반 객체 선언

```
var obj1 = const MyClass(10);
var obj2 = MyClass(10);
print('obj1 == obj2 : ${obj1 == obj2}');  // false
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare