CHAPTER 09

인터페이스

학습 목표

- 인터페이스를 사용하는 방법을 익힌다.
- 인터페이스를 생성하고 구현하는 방법을 익힌다.
- 인터페이스를 사용한 다중 상속을 이해한다.

01 다음 문장이 맞다면 O, 틀리다면 X 하시오.

① 인터페이스는 다중 상속이 가능하다. (O)

② 인터페이스는 인스턴스 변수를 가질 수 있다. (X)

③ 인터페이스는 클래스 변수를 가질 수 있다. (X)

④ 인터페이스는 속성을 가질 수 있다. (O)

⑤ 인터페이스는 클래스 메서드를 가질 수 있다. (X)

⑥ 인터페이스는 인스턴스 메서드를 가질 수 있다. (O)

02 다음 클래스와 인터페이스의 차이 중 옳지 않은 것은?

① 클래스는 class 키워드로 생성하고, 인터페이스는 interface 키워드로 생성한다.

② 클래스는 상속받았을 때 부모의 메서드를 재구현 안 해도 상관없지만, 인터페이스는 반드시 모든 메서드를 구현해야 한다.

③ 클래스는 다형성을 사용할 수 있지만, 인터페이스는 다형성을 사용할 수 없다.

④ 클래스는 하나만 상속받을 수 있지만, 인터페이스는 여러 개 상속받을 수 있다.

03 @ 기호를 붙여 만든 문자열의 특징으로 옳은 것을 고르시오.

① 이스케이프 문자를 사용할 수 없게 된다.

② 문자열을 출력할 때, 두꺼운 글씨로 출력하게 된다.

③ @ 기호를 뒤에 붙여도 된다.

④ 아무 의미가 없다.

04 IDisposable 인터페이스를 상속 받은 클래스와 함께 사용할 수 있으며, 범위를 벗어날 때 자동으로 Dispose() 메서드를 호출해주는 구문의 이름을 고르시오.

① using 구문 ② auto 구문 ③ dispose 구문 ④ release 구문

05 라디오 버튼과 체크 박스에 대한 설명으로 옳은 것을 고르시오.

① 그룹 내부에서 라디오 버튼은 한 번에 하나만 선택할 수 있다.

② 그룹 내부에서 체크 박스는 한 번에 하나만 선택할 수 있다.

③ 라디오 버튼은 기본적으로 네모난 모양이다.

④ 체크 박스는 기본적으로 동그란 모양이다.

06 다음과 같이 파일에 내용을 저장하고, 이를 읽어서 사용하는 프로그램을 작성하시오.

• 첫 번째 실행 예시

```
파일에 아무 내용도 없습니다.

> 저장할 문자열을 입력해주세요: 안녕하세요
저장했습니다.
[종료]
```

• 두 번째 실행 예시

```
이전에 입력한 내용: 안녕하세요

> 저장할 문자열을 입력해주세요: 반갑습니다
저장했습니다.
[종료]
```

• 세 번째 실행 예시

```
이전에 입력한 내용: 안녕하세요, 반갑습니다

> 저장할 문자열을 입력해주세요: 감사합니다
저장했습니다.
[종료]
```

```
static void Main(string[] args)
{
  // # 파일 읽기: 파일이 있어야 합니다.
  // 파일이 없을 때 파일을 새로 생성하는 방법은
  // 다음 장의 예외처리를 보면 아주 쉽게 구현할 수 있습니다.
  string fileInput = File.ReadAllText(@"c:\test\test.txt").Trim();
  if (fileInput.Length == 0)
  {
    Console.WriteLine("파일에 아무 내용도 없습니다.");
  }
  else
  {
    Console.WriteLine(fileInput);
  }
  Console.WriteLine();

  // # 파일 쓰기
  // AppendAllText() 메서드를 사용해보았습니다.
  // 어떤 방식을 사용해도 쓰기만 하면 상관 없습니다.
  Console.Write("> 저장할 문자열을 입력해주세요: ");
  string userInput = Console.ReadLine().Trim();
  string connector = fileInput.Length == 0 ? "" : ", ";
  File.AppendAllText(@"c:\test\test.txt", connector + userInput);
  Console.WriteLine("저장했습니다.");

  // fileInput.Length가 0일 때는 "<입력>"을 그대로 출력
  // fileInput.Length가 0이 아닐 때는 ", <입력>" 형태로 출력하게 만들었습니다.
}
```

CHAPTER 10

예외 처리

학습 목표

- 예외가 무엇인지 이해한다.
- 예외를 처리하는 기본 방법과 고급 방법을 익힌다.
- 예외 객체를 활용하는 방법을 익힌다.
- 예외를 강제로 발생시키는 방법을 익힌다.

01 다음 빈 칸을 채우시오.

① 실행 중에 발생하는 오류를 (예외)라고 부른다.

② 실행 중에 발생하는 오류를 처리하는 것을 (예외 처리)라고 부른다.

③ 프로그램이 컴파일조차 안되게 만드는 프로그래밍 언어의 문법적인 오류는 컴파일 시점 오류 또는 (문법 오류)라고 부른다.

④ catch 구문의 괄호 내부에 선언하는 변수로서 예외와 관련된 정보를 제공해주는 객체를 (예외 객체)라고 부른다.

⑤ 예외를 강제로 발생시킬 때는 (throw) 키워드를 사용한다.

02 다음 문장이 맞다면 O, 틀리다면 X 하시오.

① IndexOutOfRangeException는 컴파일 시점에 발생하는 오류이다. X

② 예외 처리는 try catch finally 구문으로만 할 수 있다. X

③ try catch finally에서 try는 반드시 있어야 하며, catch와 finally는 둘 중 하나만 넣어도 된다. O

④ return 키워드 또는 break 키워드를 만나서 try 구문을 이탈하는 경우에는 finally 구문을 실행하지 않는다. X

⑤ finally 구문 내부에서는 return 키워드를 사용할 수 없다. O

⑥ catch 구문은 예외 객체를 구분하기 위한 목적으로 여러 개 사용할 수 있다. O

⑦ 예외와 관련된 정보는 예외 객체에서 얻을 수 있다. O

⑧ 예외 객체의 자료형으로 var 키워드를 지정하면, 모든 예외를 처리할 수 있다. X

⑨ 예외 처리를 하지 않았는데 예외가 발생하면, 프로그램이 실행 중에 강제 종료된다. O

03 예외 처리 구문의 조합으로 옳지 않은 것은?

① try {} catch(Exception exception) {}

② try {} finally {}

③ try {} finally {} catch(Exception exception) {}

④ try {} catch (Exception exception) {} finally {}

04 예외를 강제로 발생시킬 때 사용하는 키워드로 옳은 것은?

① exception ② throw ③ error ④ runtime

05 다음 중에서 문법 오류가 발생하는 부분과 예외 발생이 예상되는 부분을 구분하시오.

```
/* ① */ string[] a = new string();
/* ② */ int.Parse("two");
/* ③ */ Console.WriteLine("안녕하세요"[100]);
/* ④ */ Random random = Random();
```

문법 오류: ①④
예외 발생: ②③

06 예외와 컴파일 시점 오류(문법 오류)의 차이점을 간단하게 설명하시오.

예외는 프로그램 실행 중에 발생하는 것이고, 오류는 프로그램 실행 전에 문법적인 문제로 발생하는 것이다.

CHAPTER 11

델리게이터와 람다

학습 목표

- 델리게이터, 메서드 이름, 무명 델리게이터, 람다의 관계를 이해하고 사용 방법을 익힌다.
- 델리게이터를 선언하고 사용하는 방법을 익힌다.
- 델리게이터와 관련된 연산자를 이해한다.

01 다음 빈 칸을 채우시오.

① C#은 행위를 저장하기 위해서 델리게이터와 람다라는 개념을 사용합니다.

② C#은 델리게이터를 조금 더 짧은 형태로 작성할 수 있는 (람다)라는 문법을 지원합니다.

③ 매개변수로 전달하는 메서드를 콜백 메서드라고 부른다.

02 다음 문장이 맞다면 O, 틀리다면 X 하시오.

① 정해진 형태(이름 없이) 델리게이터를 만들 수 있으며, 이를 무명 델리게이터라고 부른다. O

② 델리게이터는 자료형이므로, 클래스를 선언하는 위치와 같은 위치라면 어디든지 선언할 수 있다. O

③ 델리게이터는 클래스 외부에도 선언할 수 있다. O

④ 델리게이터에는 +, −, *, / 연산자를 사용할 수 있다. X

⑤ 스레드를 생성할 때는 Thread 클래스를 사용한다. O

03 무명 델리게이터와 람다의 기본 형식을 적으시오.

무명 델리게이터: delegate(<매개 변수>, <매개 변수>) { return <리턴> }
람다: (<매개 변수>, <매개 변수> => { return <리턴> }

연습문제

04 다음 중 델리게이터를 선언할 수 있는 위치로 옳지 않은 곳은?

```
public delegate void TestDelegateA();  ①

class Program
{
    public delegate void TestDelegateB();  ②

    static void Main(string[] args)
    {
        public delegate void TestDelegateC();  ③
        TestDelegateA delegateA;
        TestDelegateB delegateB;
    }
}
```

CHAPTER 12

Linq

학습 목표

- Linq가 무엇인지 이해한다.
- Linq의 기본 구문을 이해한다.
- Linq와 익명 객체/클래스를 함께 활용하는 방법을 익힌다.

01 다음 빈 칸을 채우시오.

① (Linq)는 컬렉션 형태의 데이터를 간단하게 리스트를 쉽게 다루고자, SQL을 본따 만든 구문입니다.

② Linq의 (where) 구문은 조건을 지정할 때 사용합니다.

③ Linq의 (orderby) 구문은 정렬을 해줍니다.

④ 클래스 이름 없이 생성하는 객체를 익명 객체라고 부른다.

⑤ (XML)(eXtensible Markup Language)은 데이터를 나타내는 데 사용되는 다목적 마크업 언어입니다.

02 다음 문장이 맞다면 O, 틀리다면 X 하시오.

① 모든 Linq 쿼리는 from, in, select 키워드를 포함한다. O

② orderby 구문의 정렬 방향(ascending 또는 descending)을 지정하지 않으면, 자동으로 descending(내림차순)이 지정된다. X

③ 모든 객체는 클래스 이름이 있어야 한다. X

④ XML의 최상위에는 요소가 하나만 올 수 있다. O

03 다음과 같은 코드를 Linq 코드로 변경하시오.

```
List<int> input = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
List<int> output = new List<int>();

foreach (var item in input)
{
    if(item < 4)
    {
        output.Add(item);
    }
}
```

```
var output = from item in input
 where item < 4
 select item;
```

04 다음과 같은 코드를 Linq 코드로 변경하시오.

```
List<int> input = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
List<int> output = new List<int>();

foreach (var item in input)
{
    if(item % 4 == 1)
    {
        output.Add(item);
    }
}
```

```
List<int> input = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
var output = from item in input
 where item % 4 == 1
 select item;
```

05 다음과 같은 코드를 Linq 코드로 변경하시오.

```
List<int> input = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
List<int> output = new List<int>();

foreach (var item in input)
{
    if (item % 4 == 1)
    {
        output.Add(item);
    }
    output.Sort();
}
```

```
List<int> input = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
var output = from item in input
 where item % 4 == 1
 orderby item
 select item;
```

06 다음과 같은 코드를 Linq 코드로 변경하시오.

```
List<int> input = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
List<int> output = new List<int>();

foreach (var item in input)
{
    if (item % 4 == 1)
    {
        output.Add(item);
    }
    output.Sort();
    output.Reverse();
}
```

```
List<int> input = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
var output = from item in input
 where item % 4 == 1
 orderby item descending
 select item;
```

07 ①~③과 같은 리스트가 있을 때, 코드들을 Linq로 변환하시오.

```
class Program
{
    class Product
    {
        public string Name { get; set; }
        public int Price { get; set; }

        public override string ToString()
        {
            return Name + " : " + Price + "원";
        }
    }

    static void Main(string[] args)
    {
        List<Product> products = new List<Product>()
        {
            new Product() { Name = "고구마", Price = 1500 },
            new Product() { Name = "사과", Price = 2400 },
            new Product() { Name = "바나나", Price = 1000 },
            new Product() { Name = "배", Price = 3000 }
        };

        var output = /* 코드 */;
    }
}
```

①
```
var output = from item in products
 orderby item.Name
 select item;
```

②
```
var output = from item in products
 where item.Price < 2000
 orderby item.Price
 select item;
```

③
```
var output = from item in products
 where item.Price < 2000
 orderby item.Price descending
 select item;
```

① products.OrderBy(item => item.Name)

② products.Where(item => item.Price < 2000).OrderBy(item => item.Price)

③ products.Where(item => item.Price < 2000).OrderBy(item => item.Price).Reverse()