컴퓨터정보과
C# 프로그래밍

11주차 예외처리와 일반화 프로그래밍

강의 순서

1. C# 환경설치 / C# 기본 구조
2. 클래스 기본(필드) + 변수,자료형
3. 클래스 기본(메소드) + 연산자,수식
4. 클래스 기본(메소드) + 제어문
5. 배열/리스트/딕셔너리
6. 클래스 기본: 접근제한자 (한정자) + 프로퍼티(속성)
7. 클래스 심화:상속
8. 클래스 심화:인터페이스/추상 클래스
9. **예외처리/일반화/…**
10. 파일처리
11. UI (Winform or WPF)
12. LINQ/Delegate/Lamda/…

복습 및 10주차 추가요소

- 다형성
  - 변수 하이딩
    - new
  - 메소드 하이딩
    - new
  - 메소드 오버라이딩
    - virtual
    - override
- 추상 클래스 (abstract class)
  - 추상 메소드를 선언할 수 있는 class
  - 참조는 될 수 있지만 인스턴스를 생성할 수 없다.
  - 파생 클래스를 통해서 인스턴스를 생성할 수 있으며
    파생 클래스는 추상 메소드의 구현(override)이 필수이다.
- 인터페이스 (interface)
  - 모든 멤버는 추상 형태 (header만 있는…)를 갖는다.
    - 구현을 해서는 안됨
  - 추상 클래스와 마찬가지로 참조는 될 수 있지만 인스턴스 생성할 수 없다.
  - 추상 클래스와 마찬가지로 파생 클래스를 통해서만 인스턴스를 생성할 수 있으며
    파생 클래스는 해당 멤버를 모두 구현해야 한다. (단, 추상클래스처럼 override는 사용하지 않음)
  - 파생 클래스에서 n개의 인터페이스를 구현(상속)할 수 있기 때문에 다중 상속을 대신해서 사용한다.

예외처리 (Exception Handling)

실행중에 발생한 오류를 처리한다.

예외 (Exception)

- 예외 : 실행시간 중에 발생한 오류 (Runtime Error)

  cf. 문법적 오류 : 컴파일 시간(문법) 오류

- 예외 처리
  - 예외가 발생했을 때 코드 상으로 처리하는 것
    (물론, 예외가 발생하지 않도록 코드 작성하는 것이 중요)
  - try – catch – finally 구문을 이용
    - try 구역 뒤에는 catch나 finally 구역이 반드시 있어야 한다.

    → try

    정상적으로 실행이 되어야 하는 구역

    → catch

    try 구문에서 예외가 발생하면 처리하는 구역

    exception 종류에 따라 구역을 여러 개 생성할 수 있음.

    → finally

    try나 catch가 끝난 후 무조건 실행하는 구역
    해당 구역은 무조건 실행하고 끝내야 하기 때문에 return을 사용할 수 없음.

- 예외 발생
  - throw new Exception();

예제

```
var name = MethodBase.GetCurrentMethod().Name;
Console.WriteLine($"=={name}============");

int a = 0;
int[] b = new int[] { 3, 4, 5, 6 };

try {
    a = int.Parse(Console.ReadLine()); //에러후보1
    //int.TryParse()
    Console.WriteLine(b[a]);//에러후보2
    //a <  b.Length
} catch (IndexOutOfRangeException ex) {
    Console.WriteLine($"입력은 0~{(b.Length - 1)} 사이 값을 넣어요!");
    return;
    Console.WriteLine(ex);//추후 파일에 저장하기
} catch (FormatException ex) {
    Console.WriteLine($"정수형 숫자를 적어주세요.");
    return;
    Console.WriteLine(ex);//추후 파일에 저장하기
} catch (Exception ex) {
    Console.WriteLine(ex.Message);
    return;
    Console.WriteLine(ex);//추후 파일에 저장하기
} finally {
    Console.WriteLine("오류발생했을까?");
}
```

일반화 프로그래밍 (generic programming)

하나의 코드에 다양한 자료형…

Generic Programming 일반화 프로그래밍

- 클래스나 메소드를 선언할 때 특정 데이터 타입을 정하지 않고 자료형 매개변수 (Type Parameter)를 통해서 적용할 수 있도록 지정하는 방식
  - 인스턴스 생성이나 메소드 호출 시 구체적인 자료형을 함께 지정해주게 되어있다.
  - 자료형은 다르지만 동작 코드가 동일한 경우 여러 개의 클래스나 메소드를 추가로 생성할 필요가 없어진다.
  - Type Parameter는 1개 이상 선언할 수 있다.
  - 이를 이용한 많은 클래스가 있다.
    - List<T>, Dictionary<Tkey, Tvalue>, Queue<T>, Stack<T>, …

예제

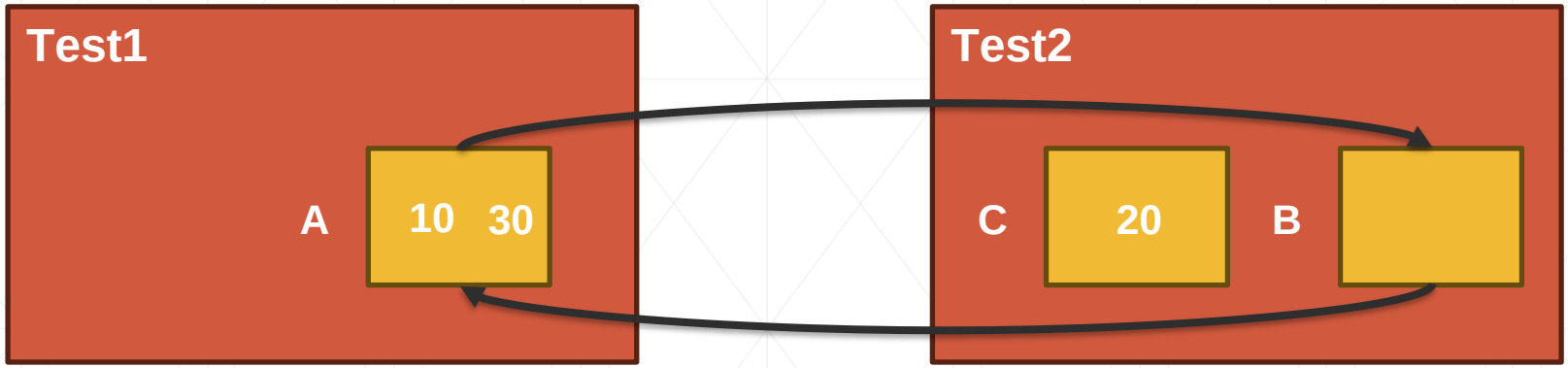
```
internal class Money<T>
{
    private int _maxCount;
    List<T> _list = new List<T>();

    참조 3개
    public Money(int maxCount)
    {
        _maxCount = maxCount;
    }

    참조 3개
    public bool Add(T money)
    {
        if (_list.Count < _maxCount) {
            _list.Add(money);
            return true;
        } else {
            return false;
        }
    }
}
```

```
static void Main(string[] args)
{
    var moneies1 = new Money<int>(5);
    var moneies2 = new Money<double>(8);
    var moneies3 = new Money<decimal>(12);

    for (int i = 0; i < 10; i++) {
        moneies1.Add(i);
        moneies2.Add(i);
        moneies3.Add(i);
    }
}
```

ref, out

지역 변수의 참조 전달

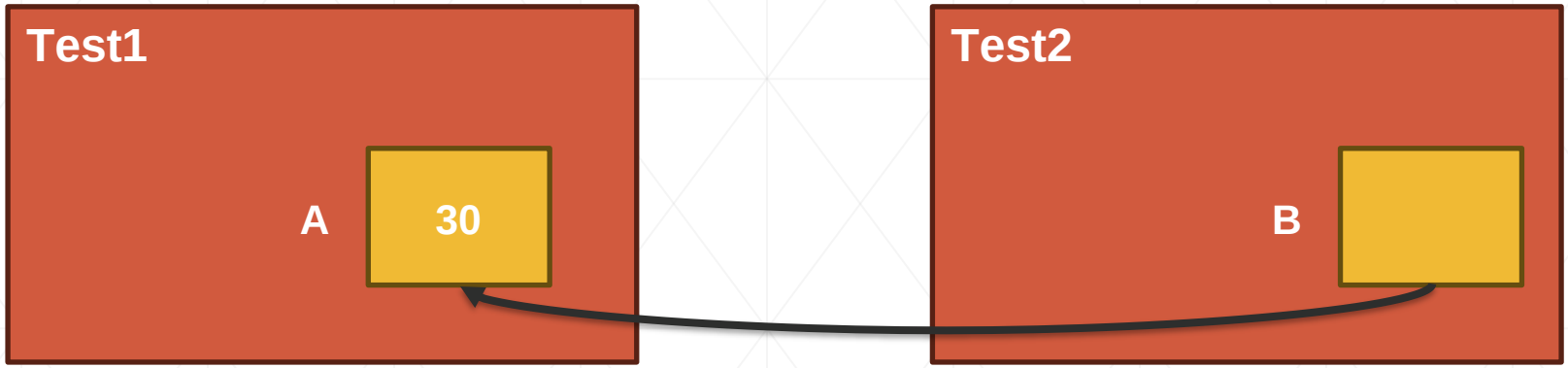
ref 매개변수




```
void Test1()
{
    int A = 10;
    Test2(ref A);
    Console.WriteLine(A);
}
```

```
void Test2(ref int B)
{
    int C = B * 2;
    Console.WriteLine(C);
    B = 30;
}
```

out 매개변수 : 출력전용 매개변수




```
void Test1()
{
    int A; //초기화 필요 없음
    Test2(out A);
    Console.WriteLine(A);
}
```

```
void Test2(out int B)
{
    int C = B * 2;
    Console.WriteLine(C);
    B = 30;
}
```

예제

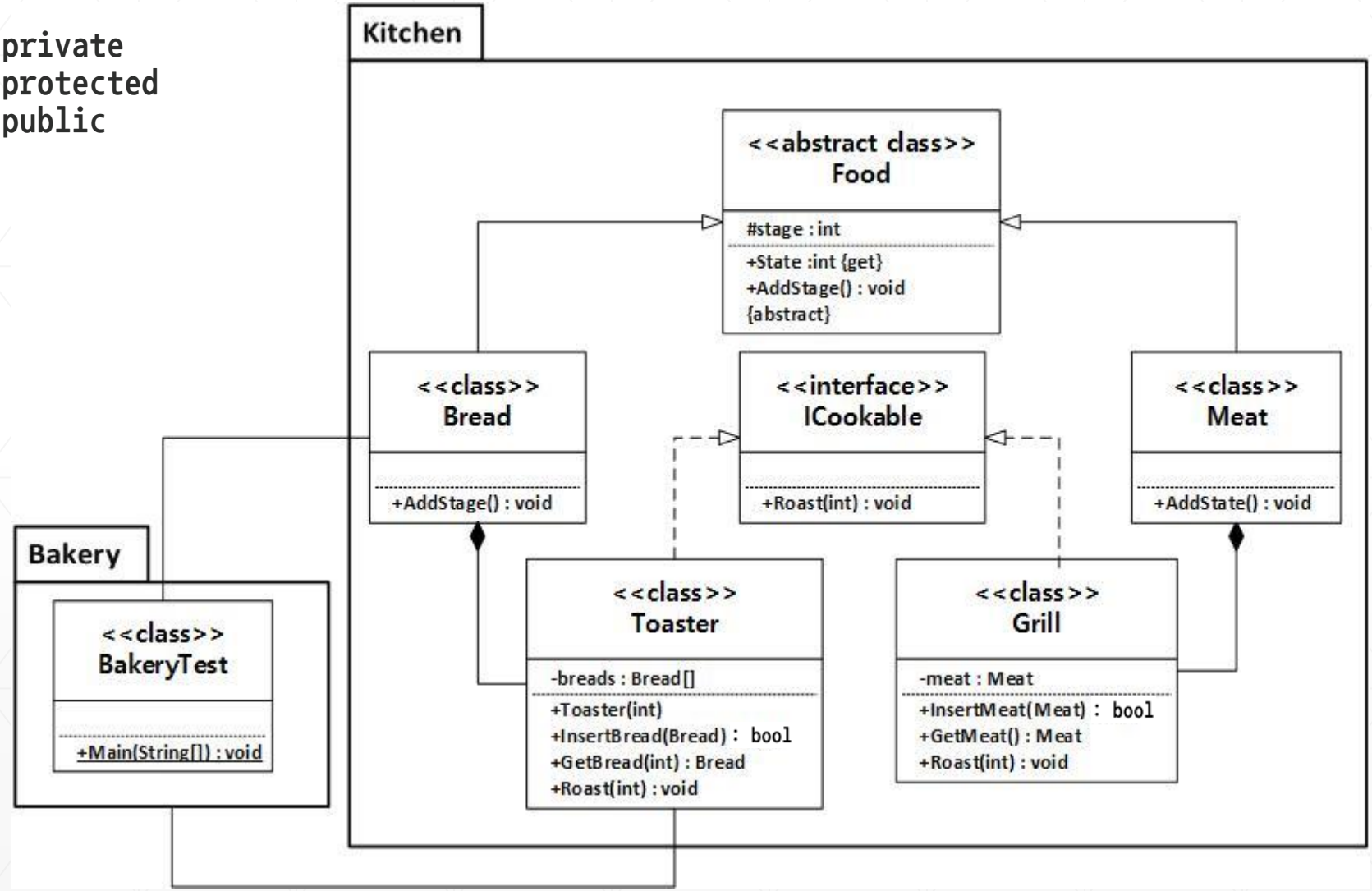
```
참조 2개
public bool Aggregate1(ref T max, ref T min)
{
    if (_list.Count > 0) {
        max = _list.Max();
        min = _list.Min();
        return true;
    } else {
        return false;
    }
}

참조 1개
public bool Aggregate2(out T max, out T min)
{
    if (_list.Count > 0) {
        max = _list.Max();
        min = _list.Min();
        return true;
    } else {
        max = default(T);
        min = default(T);
        return false;
    }
}
```

```
static void Main(string[] args)
{
    var moneies1 = new Money<int>(5);
    var moneies2 = new Money<double>(8);
    var moneies3 = new Money<decimal>(12);

    for (int i = 0; i < 10; i++)...

    int max1=0, min1 = 0;
    double max2 = 0.0, min2 = 0.0;

    moneies1.Aggregate1(ref max1, ref min1);
    moneies2.Aggregate1(ref max2, ref min2);

    Console.WriteLine($"max1:{max1} min1:{min1}");
    Console.WriteLine($"max2:{max2} min2:{min2}");

    //decimal max3, min3;
    //moneies3.Aggregate2(out max3, out min3);
    moneies3.Aggregate2(out decimal max3, out decimal min3);
    Console.WriteLine($"max3:{max3} min3:{min3}");

}
```

간이 class diagram

class diagram

- private
protected
+ public



BakeryTest

```
namespace Bakery {
    참조 0개
    class BakeryTest {
        참조 0개
        static void Main(string[] args) {
            Bread[] breads = new Bread[4];
            for (int i = 0; i < breads.Length; i++) {
                breads[i] = new Bread();
            }
            Toaster toaster = new Toaster(3);

            toaster.InsertBread(breads[0]);
            toaster.InsertBread(breads[1]);
            toaster.Roast(3);
            toaster.InsertBread(breads[2]);
            toaster.Roast(4);
            toaster.InsertBread(breads[3]);
            toaster.Roast(5);
            toaster.GetBread(1);
            toaster.Roast(3);

            foreach (var bread in breads) {
                Console.WriteLine(bread.Stage);
            }
        }
    }
}
```

내용

- **Bread**
  - AddStage()
    - 굽기정도(Stage)를 1 증가한다. 굽기 정도는 0~30 까지만 가능하다.
- **Meat**
  - AddStage()
    - 굽기정도(Stage)를 1 증가한다. 굽기 정도는 0~50 까지만 가능하다.
- **Toaster**
  - Toaster()
    - 매개변수에서 받은 만큼 _breads의 크기를 설정한다.
  - InsertBread()
    - 매개변수에서 받은 Bread인스턴스를 _breads 배열에 배치한다. 단, 비어있는 Slot에만 넣을 수 있다.
    - 배치 성공여부를 반환한다.
  - GetBread()
    - Slot의 위치를 받아 해당 Bread 인스턴스를 반환한다. 빼낸 후 Slot은 null처리한다.
  - Roast()
    - 매개변수 만큼 _breads의 인스턴스들의 Stage를 증가한다.
- **Grill**
  - InsertMeat()
    - 매개변수로 받은 Meat 인스턴스를 _meat에 배치한다. 단, _meat에 설정한 값이 없을 경우에만 할 수 있으며 배치 성공 여부를 반환한다.
  - GetMeat()
    - _meat의 Meat인스턴스를 반환한다. 반환 후 비워야 한다.
  - Roast()
    - 매개변수 만큼 _meat의 인스턴스의 Stage를 증가한다.