<<객체>>

1. 객체지향 프로그래밍이란

* 프로그램을 수많은 ‘객체’라는 기본단위로 나누고 이 객체들의 상호작용으로 서술하는 방식.
* 객체지향은 특정 언어가 아니라 개념이다.

1. 객체지향 프로그래밍의 요소
2. 캡슐화(Encapsulation)

클래스처럼 변수와 함수를 하나의 단위로 묶는 것.

1. 정보 은닉(information hiding)

프로그램의 세부 구현을 외부로 드러나지 않도록 특정 모듈 내부로 감추는 것. class에서의 public, private, protected와 같은 접근 제한이 이의 예이다.

* public: 클래스의 외부에서 사용 가능하도록 노출시키는 것이다.
* protected: 다른 클래스에게는 노출되지 않지만, 상속받은 자식 클래스에게는 노출되는 것이다.
* private: 클래스의 내부에서만 사용되며 외부로 노출되지 않는다.

1. 상속(Inheritance)

자식 객체가 부모 객체의 특성과 기능을 물려 받는 것.

1. Overriding(오버라이딩): 자식 클래스에서 상속받은 기능을 수정하여 재정의 하는 작업.
2. 다형성(Polymorphism)

하나의 변수, 또는 함수가 상황에 따라 다른 의미로 해석될 수 있는 것을 말한다.

1. 서브타입 다형성: 상위 클래스의 포인터나 참조 변수 등이 하위 클래스의 객체를 참조하게 하는 것.
2. 매개변수 다형성: 타입을 매개변수로 받아 새로운 타입을 되돌려주는 기능

템플릿, 제네릭이 이것의 예이다.

1. 임시 다형성

{1} 함수 오버로딩: 같은 이름의 함수가 매개변수에 따라 다른 기능으로 동작하도록 하는 것.

{2} 연산자 오버로딩: 기본 연산자가 해당 객체에 맞는 역할을 수행 가능하도록 하는 것.

1. 강제 다형성

{1} 묵시적 형 변환: 암묵적으로 표시하지 않아도 int에서 double과 같은 식으로 알아서 자료형이 변하는 것(자료형 승급).

{2} 명시적 형 변환: double에서 int처럼 묵시적으로 형 변환하지 않는 경우 명시적으로 표현하여 형 변환시키는 것.

<<자바>>

1. 자바 코드의 동작
2. Source code가 machine code가 되는 과정
3. Compiling a program: high level언어를 machine code로 변형(off-line)

Compiler: compiling을 해주는 프로그램

1. Assembly: assemble 언어를 machine code로 변형(off-line)

Assembler: assembly해주는 프로그램

1. Interpreter: source code를 object code로 바꿔주는 방법(on-line)
2. Byte Code

: compilation과 interpretation에서의 중간 code. Assembly code와 닮았지만 hardware에 종속적이지 않음.

* 동작 순서

1. Java program
2. Java Compiler
3. Byte-Code Program
4. Byte-Coed Interpreter(이 부분부터 Java virtual machine)
5. Machine-Language Instructions
6. Linker
7. Computer Execution of Machine-Language Instructions
8. Output of Java program
9. 자바에서의 상속

!) 자바는 다중 상속을 지원하지 않음.(C++은 지원함)

1. 상속을 왜 하는가: 중복된 코드를 쓰지 않기 위해서
2. Subclass: child class
3. Superclass: parent class
4. 자바에서의 예외처리
5. 왜 예외처리가 필요한가: 예외 처리를 사용하면 문제를 처리한 후 프로그램을 종료하지 않고 계속 실행할 수 있음.
6. Try-catch문

Try{에러 날 수 있는 code}

Catch(exception e){예외 발생시 실행할 코드}

1. 자바에서의 Polymorphism
2. Downcasting: superclass를 subclass로 형 변환(casting)시키는 것.
3. Upcasting: subclass를 superclass로 형 변환(casting)시키는 것.
4. Abstract Class(추상 클래스)

: 여러 class들 간에 비슷한 필드와 메서드를 공통적으로 추출해 만든 class

객체를 생성할 수 없고 실체 class와는 상속관계에 있음.

보통 abstract super class를 만들어 공통된 필드와 메서드를 통일할 목적으로 사용되며 유지보수성을 높이고 통일성을 유지할 수 있음. 이로 인해 실체 class구현 시 시간을 절약할 수 있고 규격에 맞게 실체 클래스를 구현할 수 있음.

1. Interface

: 개발자 사이의 코드 규약을 정하거나 여러 구현체에서 공통적인 부분을 추상화.

기본적으로 추상 메소드(구현부가 없는 메소드)의 모음이기 때문에 인터페이스를 만든다면 반드시 구현하는 클래스를 만들어야 하고 인터페이스에 명시되어 있는 추상 메소드들을 모두 구현해야 함.

인터페이스는 구현과 상속을 모두 할 수 있음. 또한 인터페이스를 사용하면 다중 상속도 가능.

1. 상속 예시
2. 그냥 클래스

Public class Car {…}

Public class Audi **extends** Car {…}

1. 추상 클래스

Public abstract class Car {…}

Public class Audi **extends** Car {…}

1. 인터페이스

Public interface Drive {…}

Public interface Park {…}

Public class Audi **implement** Drive, Park {…}

1. 자바에서의 multithreading

: 1개의 응용 프로그램을 thread라고 불리는 처리 단위를 여러 개 생성하여 같은 메모리 space안에서 복수의 처리를 병행하는 것을 의미.

1. Thread 구현 방법

{1} thread 클래스를 상속받아 구현

{2} runnable 인터페이스를 구현

1. Thread의 state

{1} created

{2} runnable

{3} Blocked

{4} Dead

1. Synchronization

: 같은 data에 여러 개의 thread가 접근할 때의 문제점을 해결하기 위한 방안.

Wait과 notify등의 함수로 접근을 제한 함. OS에서 공유메모리 관련해서 나오는 것과 비슷함.