<Report #1>

20215196 윤서은

<연습문제>

1. 소프트웨어 생명주기(software life cycle)의 6단계를 열거하고 설명하라.

1단계 : 요구 분석 (사용자의 요구 사항을 바탕으로 문제에 대해 만족 할 만한 해답이 될 수 있는 요건을 정의하는 것)

* 최선의 해결 방법을 위해 상충되는 제약 조건 사이에 존재하는 장단점과 문제점을 모두 이해하는데 도움이 됨.
* 필수 요건(반드시 만족해야 될 최소한의 요건)과 선택 요건(필수 요건에 대한 추가 사항)을 추출하여 구별해 놓아야 함.
* 모든 질문들에 대한 분석과 해결책이 고려된 후에는 공식적인 요구 명세(사용자가 원하는 문제 해답에 대한 명세의 기초가 되므로 반드시 사용자의 확인을 거쳐 문서로 작성하는 것이 중요)를 작성.

2단계 : 시스템 명세 (주어진 문제의 해답을 위해 시스템이 무엇을 수행해야 되는가를 정밀하게 정의하는 것)

* 기능적 명세(소프트웨어의 기능인 입력, 처리, 출력에 대한 명세를 포함하고 사용자와 소프트웨어 개발자간의 구체적인 합의사항 즉, 최종 해답을 명세한 것)를 통해 시스템에 대한 여러 가지 질문에 대한 대답을 정의함.
* 명세의 본질은 시스템이 무엇을 해야 되는가를 정의하는 것.

3단계 : 설계 (시스템 명세 단계에서 정의된 기능을 논리적으로 수행할 방법 즉, 소프트웨어 시스템의 전체적인 알고리즘을 설계하는 것)

* 명세에서 정의한 추상적인 무엇에서부터 구체적으로 어떻게 그것을 달성할 것인지 정의하는 것.
* 하향식 설계 방법과 상향식 설계 방법으로 나뉨.
* 하향식 설계 (= 단계별 세분화, 분할 정복) : 하나의 큰 문제를 여러 개의 작은 문제로 분해하여 이들을 각각 독립적으로 처리함으로써 전체 문제를 해결
* 상향식 설계 (객체지향 설계와 유사) : 기존에 개발되어 있는 알고리즘이나 도구를 재사용해 나가며 궁극적으로 전체 목적을 달성. 재사용함으로써 개발 비용, 기간을 단축시키고 정확성과 신뢰성을 가져올 수 있음.

4단계 : 구현 (설계된 알고리즘을 특정 프로그램 언어로 변환하는 것. 즉, 코딩하는 것)

* 프로그램 구현의 기본 원칙인 구조화 프로그래밍과 모듈러 프로그래밍 기법에 기반을 두어야 효율적이고 명확한 프로그램을 만들 수 있음.
* 구조화 프로그래밍 : 지정문, 조건문, 반복문을 이용해 프로그램을 작성하여 구조가 간단하고 제어 흐름이 명확함, 정확성 증명이 쉬움, 프로그램이 구조화 되어있어 유지 보수와 디버깅(오류 수정)이 쉬움.
* 모듈러 프로그래밍 : 프로그램을 여러 개의 모듈로 나눠 계층 형태의 관계를 갖도록 구성. 모듈이란 하나의 기능만 수행하는 서브프로그램으로서 각각 하나의 진입점, 진출점을 가짐.

5단계 : 테스트 (시스템이 설계된 대로 작동하는지 검사)

* 목적 : 소프트웨어에 내재되어 있는 오류를 찾는 것
* 테스트의 3단계 : 1) 단위검사(각 모듈별로 제공된 테스트 데이터로 검사) 2) 통합검사(몇 개의 모듈들을 그룹으로 통합시켜 검사) 3) 시스템 검사(완성된 시스템이 기능적 명세에 따라 정확하게 동작하는지 진단하기 위해 검사)
* 검수 (= 인수 검사) : 개발이 완료된 소프트웨어를 공식적으로 인수하기 전에 자체 검사를 수행하는 것 기존의 다른 경쟁 시스템과 성능을 비교하는 검사인 벤치마크 테스트와 초기 버전의 소프트웨어를 배포하여 사용하게 함으로써 검출되지 않은 오류나 결함을 찾아내는 베타 검사를 포함함.
* 이 테스트 과정은 정확성을 보장하려는 것이 아니라 확신하려는 것임. (정확성 보장을 위해서는 프로그램 증명 과정이 필요함)

6단계 : 유지 보수 (완성된 소프트웨어 시스템을 사용하기 시작한 뒤의 모 든 활동)

* 필요한 이유 : 1) 테스트 단계에서 발견하지 못한 오류나 버그가 발견되어 수리를 해야 하는 경우 2) 소프트웨어를 설계할 때의 하드웨어나 시스템 소프트웨어 그리고 실제 운영 환경이 많이 바뀌어 이에 적응시키기 위해 소프트웨어를 교정해야 되는 경우 3) 시스템 성능 향상이나 사용자의 추가 요구에 따라 소프트웨어를 갱신해야 되는 경우가 발생
* 경우에 따라서는 유지 보수비가 초기 개발비보다 더 많이 들 수 있음

3. ADT 1.1에 정의된 Natno ADT에 다음 연산을 추가하여 정의해 보라.

ADT Natno

데이타 : { i | i ∈ integer, i ≥ 0 }

연산 : for all x, y ∈ Natno

zero() ::= return 0;

isZero(x) ::= if ( x = 0 ) then return true

else return false;

succ(x) ::= return x+1;

Predecessor(x) ::= return x-1;

is\_Greater(x, y) ::= if ( x > y ) return true

else return false;

add(x, y) ::= return x+y;

substract(x, y) ::= if ( x < y ) then return 0

else return x-y;

Multiply(x, y) ::= return x\*y;

Divide(x, y) ::= return x/y;

equal(x, y) ::= if (x = y ) then return true

else return false;

End Natno

9. 객체지향 프로그래밍과 소프트웨어 재사용은 어떻게 관련되는지 예를 들어 설명하라.

객체지향프로그램은 객체 지향 방법을 사용하는데 이러한 방법을 사용하는 주요 목적 중에 하나는 소프트웨어의 재사용에 있다. 소프트웨어를 재사용하게 되면 프로그래밍 시간을 절약하고 시스템 오류를 감소시켜 신뢰성이 높은 소프트웨어를 생산할 수 있게 된다. 예를 들어 기존에 있는 클래스에 새로운 클래스를 확장시킬 때 상속을 이용한다. 재사용된 슈퍼클래스, 새로 확장된 서브 클래스가 되고 서브클래스는 슈퍼클래스의 필드나 메소드를 상속받아 재사용하게 되는 것이므로 생산 시간 단축, 비용 절감, 정확성 검증 등의 이점이 있다.

10. 메소드 오버라이딩과 메소드 오버로딩은 어떻게 다른지 설명해 보라.

메소드 오버로딩은 같은 이름의 메소드를 여러 개를 가지지만 매개변수의 유형, 개수가 다른 것이고 메소드 오버라이딩은 상위 클래스가 가지고 있는 메소드를 하위 클래스가 재정의해서 사용하는 것이다.

<프로그래밍 과제>

1. N이 1보다 큰 자연수일 때, 1부터 N까지의 합을 구하는 함수 Sum(N)과 N!을 구하는 함수 Factorial(N)의 2개의 함수를 Java 언어로 구현하라.

>>

**import** java.util.Scanner;

**public** **class** no1 {

**static** **int** Sum(**int** N) {

**int** sum = 0;

**for**(**int** i = 1; i <= N; i++) {

sum += i;

}

**return** sum;

}

**static** **int** Factorial(**int** N) {

**if**(N <= 1)

**return** 1;

**else**

**return** N \* *Factorial*(N - 1);

}

**public** **static** **void** main(String[] args) {

Scanner s = **new** Scanner(System.***in***);

System.***out***.println("1보다 큰 N의 값을 입력하세요 : ");

**int** N = s.nextInt();

System.***out***.println("1부터 " + N + "까지의 합은 " + *Sum*(N) + "입니다.");

System.***out***.println(N + "!의 값은 " + *Factorial*(N) + "입니다.");

}

}

1. N이 10, 100, 1000, 10,000, 100,000, 1,000,000, 10,000,000, 100,000,000 까지의 합인 Sum(N)을 구하여 보라. 이 때 혹시 발생되는 문제점이나 관찰되는 사실을 기술하여 보라.

>> **public** **class** no2 {

**static** **long** Sum(**int** N) {

**long** sum = 0;

**for**(**int** i = 1; i <= N; i++) {

sum += i;

}

**return** sum;

}

**public** **static** **void** main(String[] args) {

**int** a = 1;

**for**(**int** j = 1; j < 9; j++) {

a \*= 10;

System.***out***.println(*Sum*(a));

}

}

}

발생했던 문제점

* 출력문을 for문 안에 적어 출력문도 반복이 되도록 코드를 작성해야 했는데 반복문 안에 적지 않아 원하는 대로 결과값이 나오지 않았다.
* 변수의 자료형을 int로 설정했는데 숫자가 커지면서 결과값이 음수로 나오게 되어 변수의 자료형을 long으로 바꾸었더니 해결되었다. 오버플로우가 발생한 것으로 보인다. 자료형 크기를 고려하며 변수 설정을 해야겠다.

1. N을 1부터 시작해서 15까지 증가하면서 Factorial(N)의 값을 구하여   
   보라. 이 때 생기는 문제점들이나 관찰되는 사실에 대해서 기술하여   
   보라.

>>

**public** **class** no3 {

**public** **static** **int** Factorial(**int** N) {

**if**(N <= 1) {

**return** N;

}

**return** N \* *Factorial*(N - 1);

}

**public** **static** **void** main(String[] args) {

**for**(**int** i = 1; i <= 15; i++) {

System.***out***.println(*Factorial*(i));

}

}

}

발생했던 문제점

* 출력문을 적을 때   
  Factorial(i)라고 적어야 하는데 Factorial(N)이라고 적어서 원하는 결과값이 나오지 않았다.