1장. 추상 데이터 타입과 객체지향 기법

1.1 Java 입문

1.2 소프트웨어 생명 주기

1.3 추상 데이터 타입

1.4 객체지향 개념과 기법

1.5 Java 객체지향 프로그래밍

1.1 Java 입문

Java 프로그램을 실행하기 위한 준비

- JDK 다운로드
  - http://java.sun.com

- Eclipse 다운로드
  - http://www.eclipse.org

Java 프로그램

```
// 첫번째 Java 프로그램

public class JavaOne {
  static int count = 1;

  public static void main(String[] args) {
      System.out.println("The count is: " + count);
  }
}
```

기본 타입

- 8가지 기본 타입 (primitive type)
  - 정수 : int, byte, short, long
  - 실수 : float, double
  - 문자 : char (unicode 지원)

- 사용자 정의 타입 (user-defined type)
  - 클래스로 정의

상수와 변수

- 상수 (constant)
  - 값이 변하지 않는 수

- 변수 (variable)
  - 값을 저장하기 위한 기억 장소
  - 사용하기 전에 타입과 이름을 명시해야 함

터미널 입력과 출력

- 화면으로 출력
  - System.out 객체 이용
  - 예
    - System.out.println("안녕");
    - System.out.print("안녕");

- 키보드로 입력
  - System.in 객체 이용
  - 예
    - byte b = System.in.read();

제어문 - 조건문

```
if (a > b) {
   x = a;
} else {
   x = b;
}
```

```
x = (a > b) ? a : b;
```

```
switch (수식) {
   case 값1 : 코드 1;
   case 값2 : 코드 2;
   . . .
   case 값n : 코드 n;
   default  : 코드 n+1;
}
```

```
switch (ch) {
   case 'a' : x = 1;
              break;
   case 'b' : x = 2;
              break;
   default :  x = 3;
              break;
}
```

제어문 - 반복문

```
for (초기화문; 조건식; 증감문)
   명령문
```

```
while (조건식)
   명령문
```

```
do
   명령문
while (조건식);
```

```
int sum = 0;
for(int i = 0; i <= 100; i++) {
   sum = sum + i;
}
```

```
int i = 0;
int sum = 0;
while (i <= 100) {
   sum = sum + i;
   i = i + 1;
}
```

```
int i = 0;
int sum = 0;
do {
   sum = sum + i;
   i = i + 1;
} while(i <= 100);
```

클래스와 객체

- Class - 객체에 대한 명세 (변수와 메쏘드 포함)
  - 캡슐화 (encapsulation)
  - 정보은닉 (information hiding)
- Object – 클래스의 인스턴스(instance)
  - 참초 변수(reference variable) – 객체를 참조하는 변수
    - 참조 : 객체의 주소를 저장하고 있음을 나타냄

```
class Point {
    public double x; // 인스턴스 변수
    public double y;

    public void clear() {
        x = 0;
        y = 0;
    }
}
```

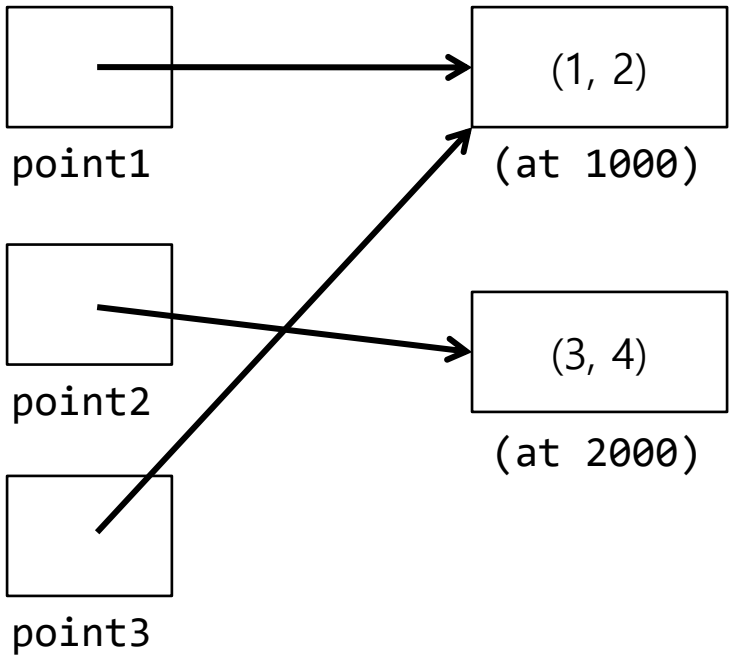
```
Point point1, point2;
point1 = new Point();
point2 = new Point();

point1.clear();
```

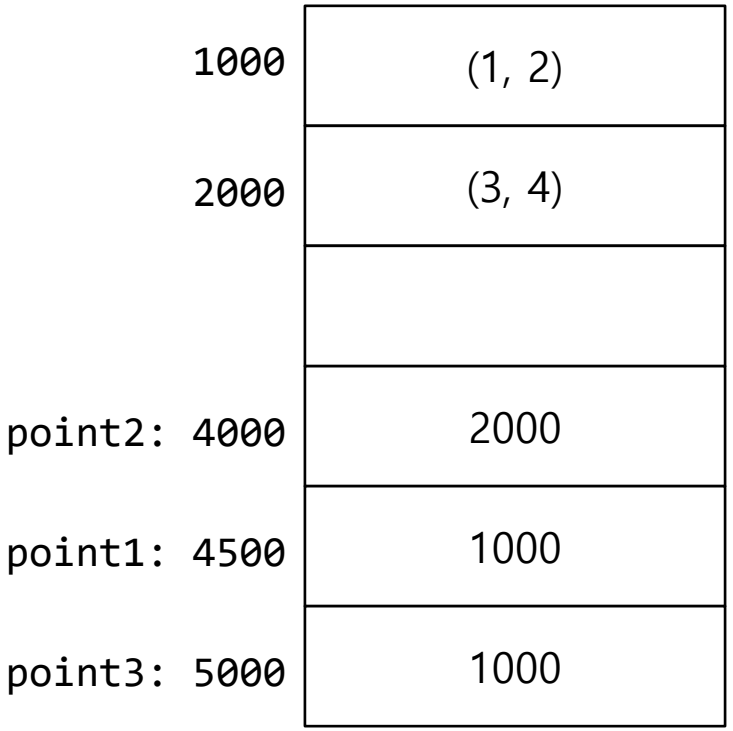
메소드 (method)

```
public class MinTest {
  public static void main(String[] args) {
      int a = 3;
      int b = 7;

      System.out.println(min(a, b));
  }

  // 메소드 min() 정의
  public static int min(int x, int y) {
      return x < y ? x : y;
  }
}
```

참조 변수 (reference variable)

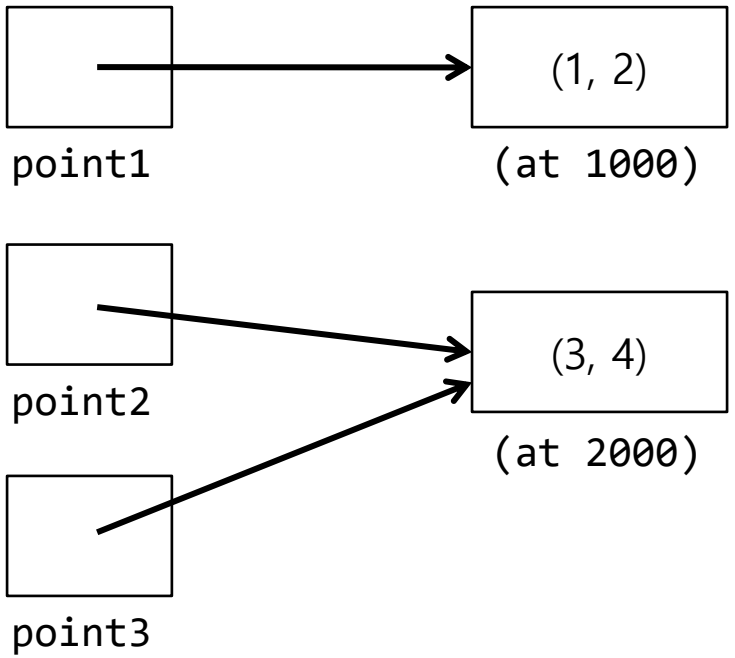
```
Point point1 = new Point();
point1.x = 1;
point1.y = 2;

Point point2 = new Point();
point2.x = 3;
point2.y = 4;

Point point3 = point1;
```

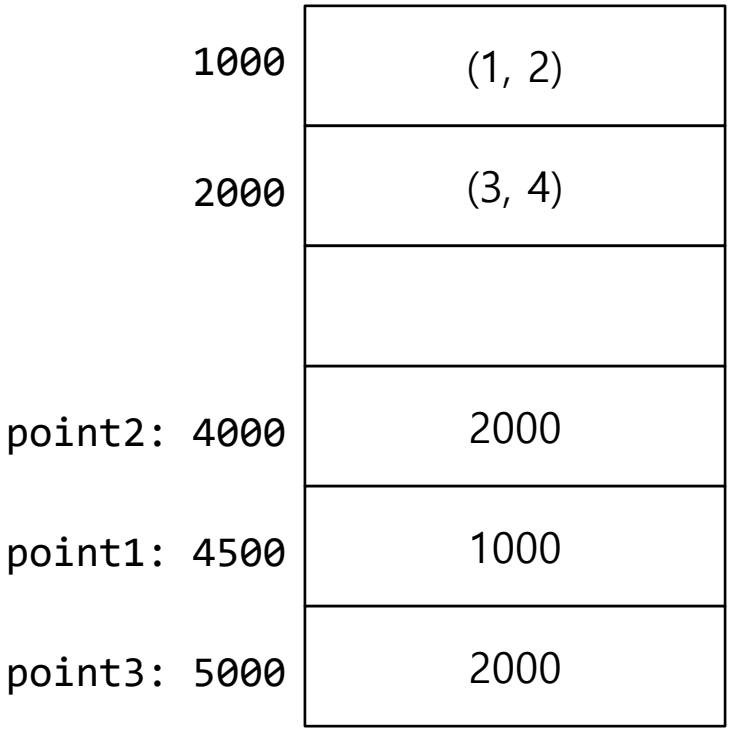




참조 변수 (reference variable)

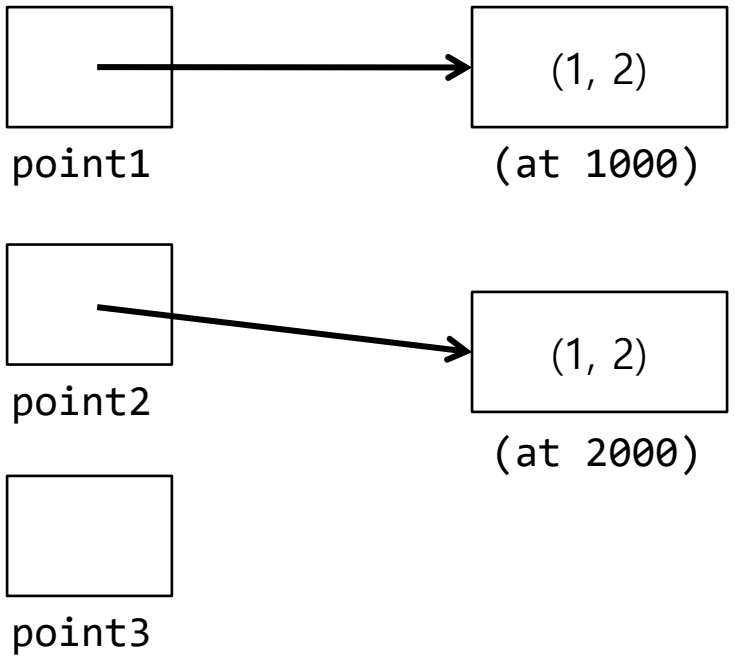
```
point3 = point1;


if (point1 == point2) {
}

if (point2 == point3) {
}
```

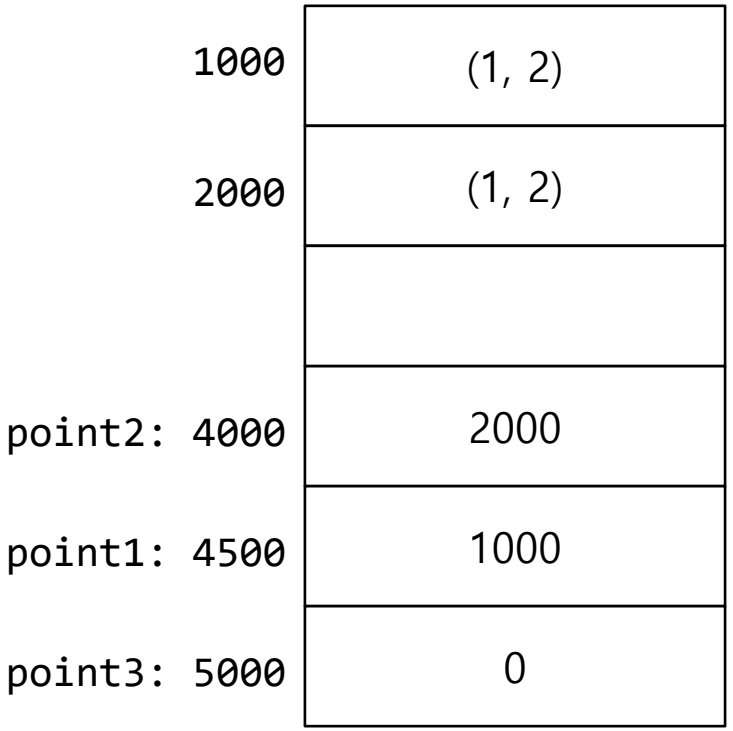




참조 변수 (reference variable)

```
point3 = null;

point2.x = 1;
point2.y = 2;

if (point1 == point2) {
}

if (point1 == point3) {
}
```





매개변수 전달 (call by value)

```
public class CallTest {
  public static void main(String[] args) {
      CallTest test = new CallTest();
      test.callByValueTest();
  }

  void callByValueTest() {
      int i = 99;
      System.out.println("before : i = " + i);
      changeValue(i);
      System.out.println("after : i = " + i);
  }

  void changeValue(int j) {
      j = 88;
  }

}
```

```
before : i = 99
after : i = 99
```

매개변수 전달 (call by referece)

```
public class CallTest {
  public static void main(String[] args) {
       CallTest test = new CallTest();
       test.callByReferenceTest();
  }

  void callByReferenceTest() {
       Test t = new Test();
       t.value = 99;
       System.out.println("before : i = " + t.value);
       changeValue(t);
       System.out.println("after : i = " + t.value);
  }

  void changeValue(Test j) {
       j.value = 88;
  }
}


class Test {
  public int value;
}
```

```
before : i = 99
after : i = 88
```

String 클래스

```
public class StringTest {
  public static void main(String[] args) {
        System.out.println("data" + " structure");
                                              // "data structure"
        System.out.println("abc" + 5);        // "abc5"
        System.out.println(5 + "abc");        // "5abc"
        System.out.println("a" + "b" + "c");  // "abc"

        System.out.println("a" + 1 + 2);              // "a12"
        System.out.println(1 + 2 + "a");              // "3a"
        System.out.println(1 + (2 + "a"));            // "12a"

        String name = "Seoul";
        int len = name.length();
        System.out.println(len);                      // 5
        char ch = name.charAt(1);
        System.out.println(ch);                       // 'e'
        String sub = name.substring(2, 4);    System.out.println(sub);
        // "ou"
  }
}
```

String 비교

```
String str1 = "a12";
String str2 = "a" + 1 + 2;

if (str1 == str2) {
    System.out.println("SAME 1");
}

if (str1.equals(str2)) {              // 내용이 같은지 비교하려면
    System.out.println("SAME 2");     // 반드시 equals 사용
}

int cmp = str1.compareTo(str2);       // 두 스트링의 대소 비교
if (cmp == 0) {
    System.out.println("같다");
} else if (cmp < 0) {
    System.out.println("작다");
} else if (cmp > 0) {
    System.out.println("크다");
}
```

배열

```
public class ArrayTest {
  public static void main(String[] args) {
        int[] array1;
        array1 = new int[100];

        int[] array3 = {3, 4, 10, 6};

        Point[] arrayOfPoints;
        arrayOfPoints = new Point[5];
        for(int i = 0; i < arrayOfPoints.length; i++)
                arrayOfPoints[i] = new Point();

        int[] a, b;
        a = new int[100];
        b = a;

        int[][] c = new int[2][3];
        c[1][1] = 0;
  }
}
```

생성자

```
public class Date {
    private int month;
    private int day;
    private int year;

    public Date() {// 무인자 생성자
        month = 1;
        day   = 1;
        year  = 2003;
    }

    public Date(int theMonth, int theDay, int theYear) {   // 3-인자 생성자
        month = theMonth;
        day   = theDay;
        year  = theYear;
    }

    public boolean equals(Object a) { // 두 Date 객체가 같은 값을 가지고 있으면
        if (!(a instanceof Date))     // true를 반환
            return false;
        Date d = (Date)a;
        return d.month == month && d.day == day && d.year == year;
    }

    public String toString() {        // Date를 스트링으로 변환
        return month + "/" + day + "/" + year;
    }
}
```

```
Date d1 = new Date();
Date d2 = new Date(4, 15, 2003);
```

static 메소드와 필드

```
System.out.println();

main 함수

String str = String.valueOf(34);

int no = Integer.parseInt("34");

int m = Math.max(2, 3);
```

```
public class Book {
    private int id;
    private static int count;

    public final static int MAX = 2147483647;
}
```

Package

- 동일한 이름을 가진 클래스가 만들어지지 않도록 하기 위함

- Java의 패키지
  - java.applet java.awt java.io
  - java.lang java.util . . .

- 사용법 1

```
java.util.Date today = new java.util.Date();
```

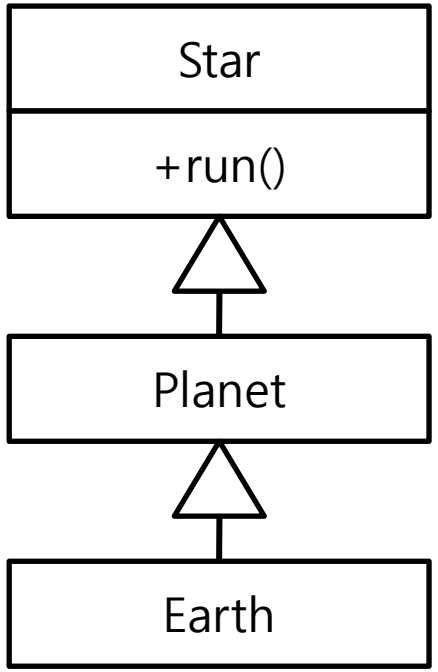
- 사용법 2

```
import java.util.Date;
import java.io.*;

Date today = new Date();
```

상속 (inheritance)

- 재사용 (인터페이스, 구현)
- 기본 클래스(base class), 파생 클래스(derived class)
- is-a 관계
  - 이행적 관계
- 서브 클래스(sub class), 슈퍼 클래스(super class)
- 접근 제어 – public, private, protected

```
class Star {
    public void run();
}

class Planet extends Star {
}

class Earth extends Planet {
}
```



추상 클래스 (abstract class)

- 추상 메소드
  - 구현은 없고 선언만 해놓은 메소드

- 추상 클래스
  - 추상 메소드를 포함하고 있는 클래스
  - 상위 클래스의 추상 메소드를 구현하지 않으면 여전히 추상 클래스임

```java
abstract class Person {
  private String name;

  abstract public void writeOutput();  // 추상 메소드

  public Person() {
       name = "no name";
  }

  public Person(String s) {
       name = s;
  }

  public void setName(String x) {
       name = x;
  }

  public String getName() {
       return name;
  }
}
```

```java
public class Student extends Person {
   private int id;

   public Student() {
        super(); // 생략 가능
        id = 0;
   }

   public Student(String s, int sno) {
        super(s);  // 생략 불가
        id = sno;
   }

   public void writeOutput() {
        System.out.println("name: " + getName());
        System.out.println("Student number: " + id);
   }

   public void reset(String x, int newId) {
        setName(x);
        id = newId;
   }

   public int getId() {
        return id;
   }

   public void setId(int sno) {
        id = sno;
   }
}
```

```java
public class StudentTest {
  public static void main(String[] args) {
        Person p1 = new Person();                // 오류
        Person p2 = new Student("tom", 10);

        System.out.println(p2.getName());
        System.out.println(p2.getId());          // 오류

        Student s = (Student)p2;
        System.out.println(s.getId());
  }
}
```

인터페이스

- 클래스
  - 인터페이스 + 구현
- 인터페이스
  - 구현없이 선언만 해놓은 메소드들의 집합

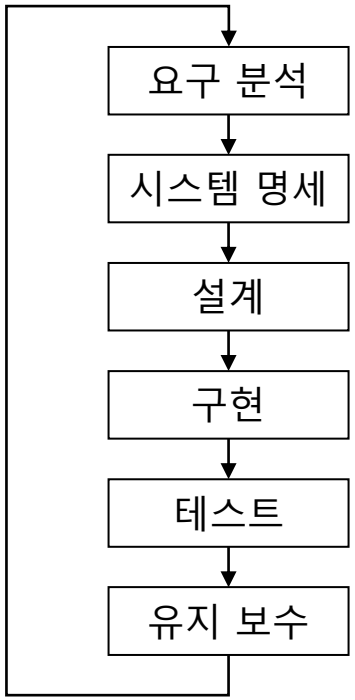
```
public interface Comparable {
   int compareTo(Comparable x);     // 클래스가 구현해야 될 추상 메소드
}
```

```
public interface Comparable {
  int compareTo(Comparable x);
}

public class Height implements Comparable {
  private int value;

  public Height(int x) {
       value = x;
  }

  public String toString() {
       return Integer.toString(value);
  }

  public int compareTo(Comparable x) {
       return value < ((Height)x).value ? -1 :
                value == ((Height)x).value ? 0 : 1;
  }

  public static void main(String[] args) {
       Height h1 = new Height(1);
       Height h2 = new Height(1);
       Height h3 = new Height(2);

       System.out.println(h1.compareTo(h2));      // 0
       System.out.println(h1.compareTo(h3));      // -1
       System.out.println(h3.compareTo(h1));      // 1
  }
}
```

1.2 소프트웨어 생명주기

소프트웨어 생명 주기

- 소프트웨어의 중요 조건
  - 효율성
  - 정확성



1.3 추상 데이터 타입

데이타의 추상화

- 크고 복잡한 문제의 해결 방법은 추상화

- 추상화 (abstraction)
  - 자세한 것은 무시하고 필수적이고 중요한 속성만을 골라서 **단순화**시키는 과정

- 데이타 추상화 (data abstraction)
  - 프로그램 속의 복잡한 데이타에 추상화 개념을 적용하여 **단순화**
  - 기존의 잘 정의된 개념들을 이용하여 표현

데이타 vs 데이타 타입

- 데이타
  - 프로그램의 처리 대상이 되는 모든 것
  - 특별히 값 (value) 자체를 의미하기도 함

- 데이타 타입
  - 데이타의 집합과 이 데이타에 적용할 수 있는 연산의 집합
  - 종류
    - 시스템 정의 (system-defined) 데이타 타입
    - 사용자 정의 (user-defined) 데이타 타입
  - 예) integer 시스템 정의 데이타 타입
    - 데이타 : 정수 (... –2, -1, 0, 1, 2 ...)
    - 연산자 : +, -, *, /

추상 데이타 타입

- 추상 데이타 타입 (abstract data type: ADT)
  - 데이타 타입의 논리적 정의

  - **데이타와 연산의 본질에 대한 명세만 정의**한 데이타 타입
    - 데이터 (변수, 필드 field)
    - 연산 (함수, 메소드 method)

자연수(Natno) 추상 데이타 타입

```
ADT Natno
   데이타 : { i ¦ i ∈ integer, i ≥ 0}

   연산자 : for all x, y ∈ Natno

       zero() ::= return 0;→ Natno
       isZero(x) ::= if (x) then return false
                     else return true;
       succ(x) ::= return x+1;
       add(x, y) ::= return x+y;
       subtract(x, y) ::= if (x<y) then return 0
                          else return x-y;
       equal(x, y) ::= if (x=y) then return true
                       else return false;

End Natno
```

1.4 객체지향 개념과 설계

객체지향 개념과 설계

- 객체지향 설계(object-oriented design) 방법
  - 의미
    - 객체지향 개념을 기초로 잘 정의된 객체들을 먼저 식별한 후 이들이 상호 작용하게 구성함으로써 원하는 결과를 생성하게 하는 것
  - 방법
    - 기초적이고 핵심적인 객체를 식별하고 설계한 뒤에 시스템의 기능적 분해를 고려
    - 상향식 설계(bottom-up design) 방법과 비슷

- 객체지향 개념의 본질
  - 캡슐화 (encapsulation)
  - 상속 (inheritance)
  - 다형성 (polymorphism)

캡슐화 (encapsulation)

- 객체(object)
  - public, private, protected
- 캡슐화(encapsulation)
- 정보 은닉 (information hiding)

상속 (inheritance)

- 상속(inheritance)
- is-a 관계
- 메소드 오버라이딩(method overriding)
- 코드의 재사용 (reuse)
  - 프로그래밍 시간 절약
  - 검증된 클래스를 재사용함으로써 신뢰성 높은 소프트웨어 생산

동적 바인딩 (dynamic binding)

- 실행시간에 수행할 코드 결정

```
class Person {
    public void output() {
        System.out.println("Person's object method");
    }
}

class Student extends Person {
    public void output() {
        System.out.println("Student's object method");
    }
}
```

- 실행 코드와 실행결과

```
Person p = new Person();
p.output();
p = new Student();
p.output();
```

```
Person's object method
Student's object method
```

1.5 Java 객체지향 프로그래밍

```java
public class Rectangle {
   private int x, y;                // 사각형의 x, y 좌표
   private int width, height;       // 사각형의 가로, 세로

   public Rectangle() {
        x = y = width = height = 0;
   }

   public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
   }

   public void draw() {
        System.out.println("Rectangle: " + x + ", " + y + ", "+ width + ", " + height);
   }

   public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle(0, 8, 100, 150);

        r1.draw();
        r2.draw();
   }
}
```

Rectangle 프로그램 실행

- Rectangle Java 소스 프로그램을 Rectangle.java라는 이름의 화일로 저장

- Java 컴파일러(Javac)로 컴파일
  - **C:> javac Rectangle.java**

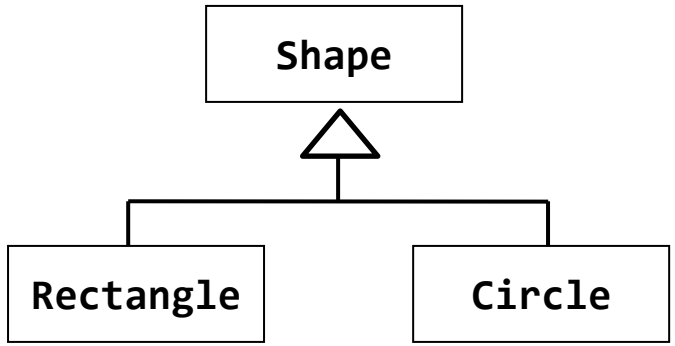
- bytecode 번역기(java)로 실행
  - **C:>java Rectangle**

- 실행 결과

  Rectangle: 0, 0, 0, 0

  Rectangle: 0, 8, 100, 150

```
abstract class Shape {
   protected Color color;
   protected int x;
   protected int y;

   protected Shape(Color c, int x, int y) {
         color = c;
         this.x = x;
         this.y = y;
   }

   public abstract void draw();
}
```



```
class Rectangle extends Shape {
   private int width;
   private int height;

   public Rectangle(Color c, int x, int y, int width, int height) {
         super(c, x, y);

         this.width = width;
         this.height = height;
   }

   public void draw() {
         System.out.println("Rectangle: " + x + ", " + y + ", " + width + ", " + height);
   }
}
```

```java
class Circle extends Shape {
   private int radius;

   public Circle(Color c, int x, int y, int raduis) {
        super(c, x, y);

        this.radius = radius;
   }

   public void draw() {
        System.out.println("Circle: " + x + ", " + y + ", " + radius);
   }
}
```

```java
public class GraphicsProgram {
   public static void main(String[] args) {
        Shape s1 = new Rectangle(Color.red, 0, 5, 200, 300);
        Shape s2 = new Circle(Color.green, 20, 30, 100);

        s1.draw();          // Rectangle: 0, 5, 200, 300
        s2.draw();          // Circle: 20, 30, 0
   }
}
```