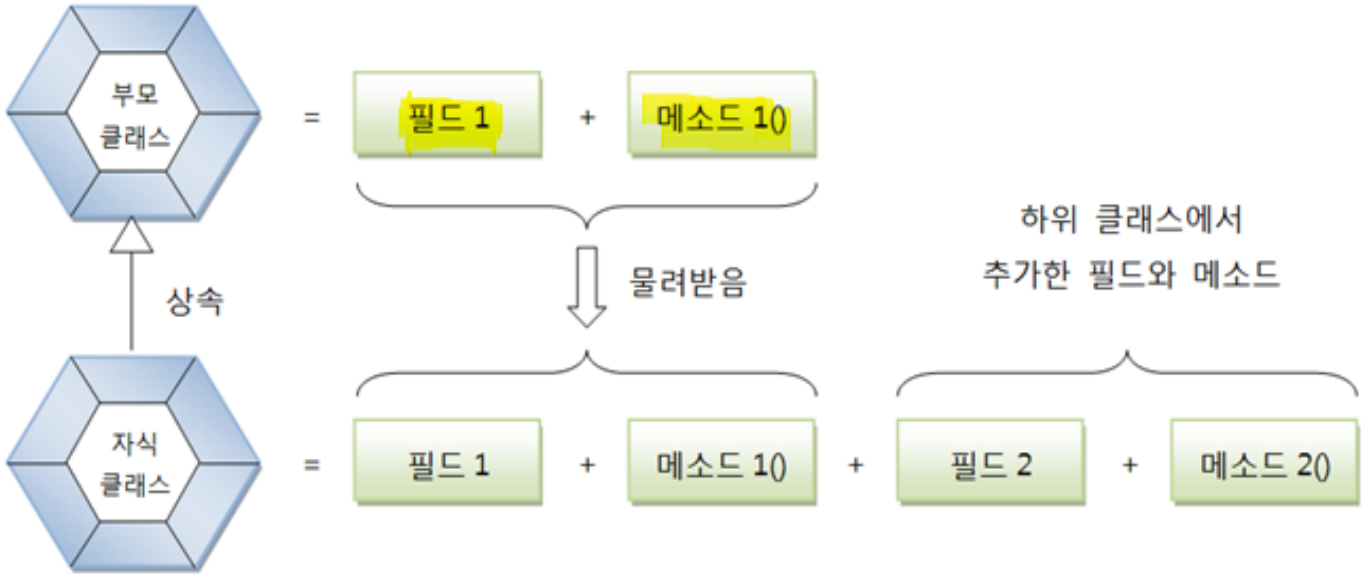
상속

안 화 수

상속

- 상속(Inheritance)
  1. 부모(상위, 슈퍼) 클래스의 멤버를 자식(하위, 서브, 파생) 클래스가 물려 받는 것.
  2. 부모 클래스의 멤버 중에서 필드와 메소드만 자식 클래스로 상속 된다.
  3. 부모 클래스의 멤버 중에서 생성자는 자식 클래스에 상속되지 않는다.



상속

- 상속(Inheritance)의 장점

1. 부모 클래스를 재사용해서 자식 클래스의 개발속도가 빨라진다.

2. 반복되는 코드의 중복을 줄일 수 있다.

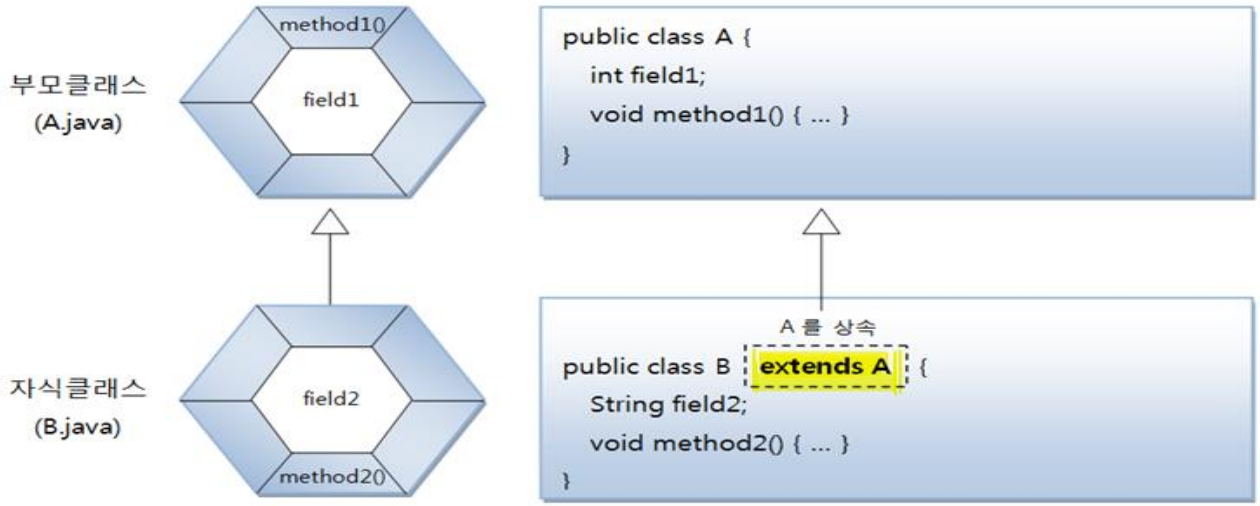
3. 유지 보수가 용이하다.

4. 객체의 다형성 구현이 가능하다.

상속

클래스의 상속

- 부모 클래스를 상속 받기 위해서는 extends로 상속 받는다.



- 자바는 1개의 클래스만 상속받는 클래스의 단일 상속만 가능하다.

```
class 자식클래스 extends 부모클래스 1, 부모클래스 2 {
}
```

상속

- 자바 api 클래스의 상속 예1.

```
public class FrameTest {

    private Frame f;

    public FrameTest() {
        f = new Frame( "Frame Test" );
        f.setSize( 400, 300 );
        f.setLocation( 100, 100 );
        f.setBackground( Color.green );
        f.setVisible( true );
        f.setResizable(false);

        f.addWindowListener( new WindowAdapter() {
            public void windowClosing( WindowEvent e ) {
                System.exit( 0 );
            }
        });

    }//생성자 end

    public static void main( String[] args ) {
        FrameTest ft = new FrameTest();
    }
}
```

상속

- 자바 api 클래스의 상속 예2.

```
public class FrameTestEx extends Frame {

    public FrameTestEx() {

        //  f = new Frame( "Frame Test" );
        super("Frame Test");  // 부모 클래스의 생성자 호출

        addWindowListener( new WindowAdapter() {
            public void windowClosing( WindowEvent e ) {
                System.exit( 0 );
            }
        });

        setSize( 400, 300 );
        setLocation( 100, 100 );
        setBackground( Color.green );
        setVisible( true );
    }//생성자 end

    public static void main( String[] args ) {
        FrameTestEx ft = new FrameTestEx();
    }
}
```

상속

- 사용자 정의 클래스의 상속 예 (1/2)

```
class Point2D{                          // 부모 클래스
  private int x;                        // 필드
  private int y;

  public int getX( ){                   // 메소드
     return x;
  }
  public void setX(int new_X){
    x=new_X;
  }
  public int getY( ){
     return y;
  }
  public void setY(int new_Y){
    y=new_Y;
  }
}

class Point3D extends Point2D{          // 자식 클래스
  private int z;
  public int getZ( ){
     return z;
  }
  public void setZ(int new_Z){
    z=new_Z;
  }
}
```

상속

- 사용자 정의 클래스의 상속 예 (2/2)

부모 클래스의 필드와 메소드는 자식 클래스에서 상속을 받아서 사용할 수 있다.

```
public class SuperSub00{

   public static void main(String[] args){

      Point3D  pt=new Point3D( );
      pt.setX(10); //상속받아 사용
      pt.setY(20); //상속받아 사용
      pt.setZ(30); //자신의 것 사용
      System.out.println( pt.getX( )//상속받아 사용
                        +", "+ pt.getY( )   //상속받아 사용
                        +", "+ pt.getZ( ));//자신의 것 사용
   }

}
```

상속-필드

- 상속에서의 필드

부모 클래스의 필드 x, y는 자식 클래스로 상속된다.

```
class Point2D{                          // 부모 클래스
  protected int x=10;
  protected int y=20;
}
class Point3D extends Point2D{ // 자식 클래스
  protected int z=30;
  public void print( ){
    System.out.println(x +", "+ y +", "+z);    //x와 y는 상속 받아 사용하는 필드
  }                                                 // 10, 20, 30
}

public class SuperTest02{
  public static void main(String[] args){
    Point3D pt=new Point3D( );
    pt.print( );
  }
}
```

상속-필드

- 상속에서의 필드

자식 클래스에서 x, y를 재 정의하면 부모 클래스의 x, y는 은닉변수가 된다.

```
class Point2D{                              // 부모 클래스
  protected int x=10; //자식 클래스에서 x,y를 재정의하면 부모 클래스의 x,y는 은닉변수가 된다.
  protected int y=20; //은닉 변수 or 쉐도우 변수
}
class Point3D extends Point2D{              // 자식 클래스
  protected int x=40; //부모 클래스에 존재하는 멤버변수를 자식 클래스에 다시 한 번 정의함
  protected int y=50;

  protected int z=30;
  public void print(){
    System.out.println(x +", "+ y +", "+z);   //자식 클래스의 재정의된 x,y가 출력된다.
  }                                           // 40, 50, 30
}

public class SuperTest03{
  public static void main(String[] args){
    Point3D pt=new Point3D();
    pt.print();
  }
}
```

상속-필드

- 상속에서의 필드

부모 클래스의 은닉된 x, y를 자식 클래스에서 접근 하기 위해서는 super를 이용한다.

```
class Point2D{                          // 부모 클래스
  protected int x=10; //은닉 변수
  protected int y=20;
}
class Point3D extends Point2D{          // 자식 클래스
  protected int x=40; //부모 클래스에 존재하는 멤버변수를 자식 클래스에 다시 한 번 정의함
  protected int y=50;

  protected int z=30;
  public void print(){
    System.out.println(x +", "+ y +", "+z);    // 자식 클래스의 재정의된 x,y가 출력
  }                                            // 40, 50, 30
  public void print02(){
    System.out.println(super.x+", "+super.y+", "+z); // super를 이용해서 부모 클래스의 x,y출력
  }                                                  // 10, 20, 30
}
public class SuperTest04{
  public static void main(String[] args){
    Point3D pt=new Point3D();
    pt.print();
    pt.print02();
  }
}
```

상속-메소드

- 상속에서의 메소드

부모 클래스의 메소드는 자식 클래스로 상속된다.

부모 클래스에서 자식 클래스의 메소드는 접근 할 수 없다.

```
class Parent {                              // 부모 클래스
    public void parentPrn() {
        System.out.println("부모 클래스의 메서드는 상속된다.");
    }
}
class Child extends Parent {                // 자식 클래스
    public void childPrn() {
        System.out.println("자식 클래스의 메서드는 부모가 사용할 수 없다.");
    }
}
public class SuperTest05 {
    public static void main(String[] args) {
        Child c = new Child();
        c.parentPrn();          // 상속받은 parentPrn() 호출
        c.childPrn();

        Parent p = new Parent();
        p.parentPrn();          // 부모 클래스 자기 자신의 parentPrn() 메소드 호출
        // p.childPrn( );       // 부모 클래스에서 자식 클래스의 메소드에 접근할 수 없다.
    }
}
```

상속-메소드

- 메소드 재정의(Method Overriding)
  부모 클래스의 메소드를 자식 클래스에서 재정의(overriding)해서 사용하는 것

- 메소드 재정의 방법
  1. 부모 클래스의 메소드 이름과 형식을 동일하게 사용해야 한다.
  2. 접근 제한을 더 강하게 정의 할 수 없다.
     public을 default나 private으로 수정 불가능
     반대로 default는 public 으로 수정 가능
  3. 새로운 예외를 throws 할 수 없다.

- 메소드 재정의(Method Overriding) 결과
  1. 자식 클래스에서 부모 클래스의 메소드를 재정의하면, 자식 클래스에서 메소드를 호출하면 재정의된 메소드만 호출되며 부모 클래스의 메소드는 은닉 되어서 더 이상 사용할 수 없다.
  2. 자식클래스에서 부모 클래스의 은닉 메소드를 사용하기 위해서는 super를 이용한다.

상속-메소드

- 메소드 재정의(Method Overriding)

자식 클래스에서 메소드를 호출하면, 메소드 오버라이딩된 메소드만 호출된다.

```
class Parent {                                  // 부모 클래스
    public void parentPrn() {
        System.out.println("부모 클래스 : ParentPrn 메서드");
    }
}
class Child extends Parent {                    // 자식 클래스
    public void parentPrn() {    // 메소드 오버라이딩
        System.out.println("자식 클래스 : ParentPrn 메서드");
    }
    public void childPrn() {
        System.out.println("자식 클래스 : ChildPrn 메서드");
    }
}
public class SuperTest06 {
    public static void main(String[] args) {
        Child c = new Child();
        c.parentPrn();   // 재정의된 자식 클래스의 메서드 호출
        c.childPrn();

        Parent p = new Parent();
        p.parentPrn();   // 부모 클래스 자기 자신의 메서드 호출
    }
}
```

상속-생성자

- 상속에서의 생성자

1. 생성자는 기본적으로 상속이 되지 않는다

2. 자식클래스를 이용해서 객체를 생성할 때 자식클래스의 생성자(기본생성자, 매개변수 있는 생성자 모두)가 호출되면, 부모클래스의 기본생성자가 자동으로 호출된다.

3. 매개변수가 있는 생성자가 있을 경우에는 더이상 컴파일러가 기본 생성자를 자동으로 생성해 주지 않는다.

4. 부모 클래스의 매개변수가 있는 생성자를 자식 클래스에서 호출할때는 super()를 이용해서 호출할 수 있다.
   단, super()는 자식 클래스의 생성자 안에서만 사용 해야 한다.

상속-생성자

- 상속에서의 생성자

생성자는 상속되지 않으며, 자식 클래스의 생성자(기본생성자, 매개변수가 있는 생성자)가 호출되면 부모 클래스의 기본생성자는 연쇄적으로 자동으로 호출된다.

```
class Point2D {                              // 부모 클래스
    protected int x = 10;
    protected int y = 20;
    public Point2D() {                       // 기본 생성자
        System.out.println("부모 클래스인 Point2D 생성자 호출");
    }
}
class Point3D extends Point2D {              // 자식 클래스
    protected int z = 30;
    public void print() {
        System.out.println(x + ", " + y + "," + z);
    }
    public Point3D() {                       // 기본 생성자
        System.out.println("자식 클래스인 Point3D 생성자 호출");
    }
}
public class SuperTest07 {
    public static void main(String[] args) {
        Point3D pt = new Point3D(); // 생성자 호출
        pt.print();
    }
}
```

상속-생성자

- super

  부모 클래스를 의미하는 내부 레퍼런스 변수

- super.

  부모 클래스의 은닉된 필드와 은닉된 메소드를 호출할 때 사용한다.

  super.x

  super.parentPrn()

- super()

  1. super()는 부모클래스의 매개변수를 가진 생성자를 호출할 때 사용한다.
  2. super()는 자식클래스의 생성자 안의 첫번째 라인에서 사용해야 한다.
  3. super()를 이용해서 부모클래스의 매개변수를 가진 생성자를 호출하면, 더이상 부모클래스의 기본 생성자를 호출해주지 않는다.

상속-생성자

- 상속에서의 생성자

부모 클래스의 매개변수가 있는 생성자는 super()를 이용해서 직접 호출해야 한다.

```
class Point2D {                                    // 부모 클래스
    protected int x = 10;
    protected int y = 20;
    public Point2D() {                             // 기본 생성자
        System.out.println("부모 클래스인 Point2D 생성자 호출");
    }
    public Point2D(int xx, int yy) {
        x = xx;              // x = 50
        y = yy;              // y = 60
    }
}
class Point3D extends Point2D {                    // 자식 클래스
    protected int z = 30;
    public void print() {
        System.out.println(x + ", " + y + "," + z);
    }
    public Point3D() {
        super(50, 60);   // 부모클래스의 매개변수가 있는 생성자 호출
        System.out.println("자식 클래스인 Point3D 생성자 호출");
    }
}
public class SuperTest08 {
    public static void main(String[] args) {
        Point3D pt = new Point3D(); // 생성자 호출
        pt.print();
    }
}
```

레퍼런스 형변환

레퍼런스 형변환

두개의 클래스 사이에 상속관계가 있어야 레퍼런스 형변환이 가능하다.

- 자동 형변환(업 캐스팅)
  1. 자식클래스에서 부모클래스로 형변환 하는 것이다.
  2. 참조 가능한 영역이 축소가 된다.
  3. 컴파일러에 의해서 자동 형변환이 된다.

- 강제 형변환(다운 캐스팅)
  1. 부모클래스에서 자식클래스로 형변환 하는 것이다.
  2. 참조 가능한 영역이 확대가 된다.
  3. 컴파일러에 의해서 자동 형변환이 되지 않기 때문에 프로그래머가 직접 강제 형변환을 해야 하며, 강제 형변환시 자료형을 생략할 수 없다.

레퍼런스 형변환

- 레퍼런스 형변환 : 자동 형변환(업캐스팅)

```
class Parent {                          // 부모 클래스
    public void parentPrn() {
        System.out.println("슈퍼 클래스 : ParentPrn 메서드");
    }
}
class Child extends Parent {            // 자식 클래스
    public void childPrn() {
        System.out.println("서브 클래스 : ChildPrn 메서드");
    }
}
public class RefTest01 {
    public static void main(String[] args) {
        Child c = new Child();
        c.parentPrn();                  // 상속받은 메소드 호출
        c.childPrn();

        Parent p;
        p = c;                          // 업캐스팅(자동 형변환)

        Parent p1 = new Child();        // 업캐스팅(자동 형변환)

        p.parentPrn();          // 업 캐스팅 후에는 부모로부터 상속받은 메서드만 호출할 수 있다.
//      p.childPrn();           // 자식 클래스의 메소드는 접근 할 수 없다.
    }
}
```

레퍼런스 형변환

- 레퍼런스 형변환 : 강제 형변환(다운캐스팅)

```
class Parent03 {                                    // 부모 클래스
    public void parentPrn() {
        System.out.println("슈퍼 클래스 : ParentPrn 메서드");
    }
}
class Child03 extends Parent03 {                    // 자식 클래스
    public void childPrn() {
        System.out.println("서브 클래스 : ChildPrn 메서드");
    }
}
public class RefTest03 {
    public static void main(String[] args) {
        Parent03 p = new Child03();                 // 자동 형변환(업캐스팅)
        p.parentPrn();                              // 부모가 상속해준 메소드만 호출 가능함.
//      p.childPrn();                               // 자식 클래스의 메소드에 접근할 수 없다.

        Child03 c;
        c = (Child03) p;                            // 강제 형변환으로 다운 캐스팅

        Child03 c1 = (Child03) p;                   // 강제 형변환(다운 캐스팅)
        c.parentPrn();                              // 상속 받은 메소드 호출 가능
        c.childPrn();                               // 자식 클래스의 메소드 호출 가능
    }
}
```

접근 제한자

- 상속에서의 접근 제한자

접근제어자	자신의클래스	같은패키지	다른패키지 (같은패키지)	하위클래스 (같은패키지)	하위클래스 (다른패키지)
private	O	X	X	X	X
생략(default)	O	O	X	O	X
protected	O	O	X	O	O
public	O	O	O	O	O

접근 제한자

- 상속에서의 접근 제한자

1. 상속 관계가 있는 경우 : 같은 패키지
   2개의 클래스(부모,자식 클래스)가 같은 패키지 안에 들어 있을 때는 부모의 접근제어자가 default, protected, public 접근제한자인 경우에 자식클래스에서 접근 할 수 있다.
   단, private접근 제어자만 자식 클래스에서 접근 할 수 없다.

2. 상속 관계가 있는 경우 : 다른 패키지
   2개의 클래스(부모,자식 클래스)가 다른 패키지 안에 들어 있을 때는 부모의 접근제한자가 protected, public 접근제한자인 경우에 자식클래스에서 접근 할 수 있다.

3. 상속 관계가 없는 경우 : 다른 패키지
   2개의 클래스가 서로 다른 패키지 안에 들어 있을 때는 public 접근제한자로 되어 있어야만 다른 클래스에서 접근 할 수 있다.

접근 제한자

- 상속에서의 접근 제한자

src - packTest - packOne - AccessTest.java (부모클래스)

```
private      int a=10;
             int b=20;
protected    int c=30;
public       int d=40;
```

p- SubOne.java (자식클래스) : protected, public만 접근가능

p - SuperSubA.java (제3의 클래스) : public만 접근가능

접근 제한자

- 상속에서의 접근 제한자 : AccessTest.java (1/2)

```
package packTest.packOne;

public class  AccessTest {   //부모 클래스
   private int     a=10;     //[1] private
   int             b=20;     //[2] 기본 접근 지정자
   protected  int  c=30;     //[3] protected
   public     int  d=40;     //[4] public

   public void print( ){
      System.out.println("AccessTest의 print");
      System.out.println(a);
      System.out.println(b);
      System.out.println(c);
      System.out.println(d);
   }

}
```

접근 제한자

- 상속에서의 접근 제한자 : SuperSubA.java (2/2)

```
import packTest.packOne.AccessTest;

class SubOne extends AccessTest { //자식 클래스
  void subPrn( ){
    System.out.println(a); //[1. Sub] private-X
    System.out.println(b); //[2. Sub] 기본 접근 지정자-X
    System.out.println(c); //[3. Sub] protected-O
    System.out.println(d); //[4. Sub] public-O
  }
}

public class SuperSubA{
  public static void main(String[] args){
    AccessTest at=new AccessTest();
    at.print();
    System.out.println("main");
    System.out.println(at.a); //[1. main] private-X
    System.out.println(at.b); //[2. main] 기본 접근 지정자-X
    System.out.println(at.c); //[3. main] protected-X
    System.out.println(at.d); //[4. main] public-O
  }
}
```