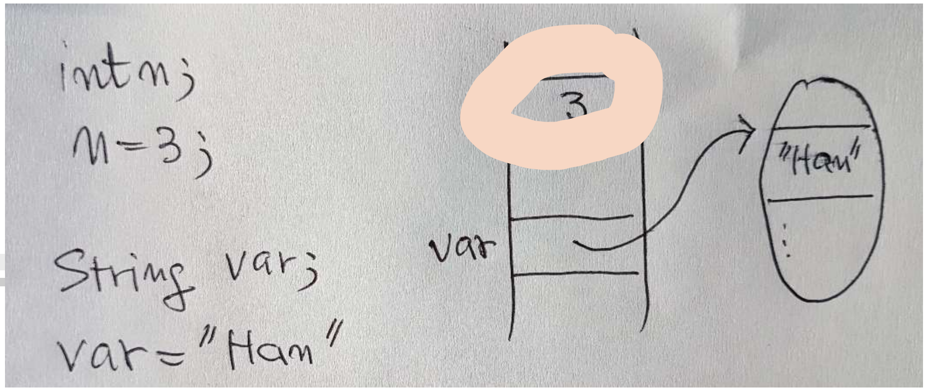
5.3 OBJECTS and REFERENCE

- Variable of a class type
  - » a variable of a class type are names for objects of their class
  - » a variable of a class type can name an object
- Variables of the primitive type (int, double, and char)
  - » ex) number 6 can be the value of a variable of type

Variables: Class Type vs. Primitive Type



```
int n;
n=3;

String var;
var="Ham"
```

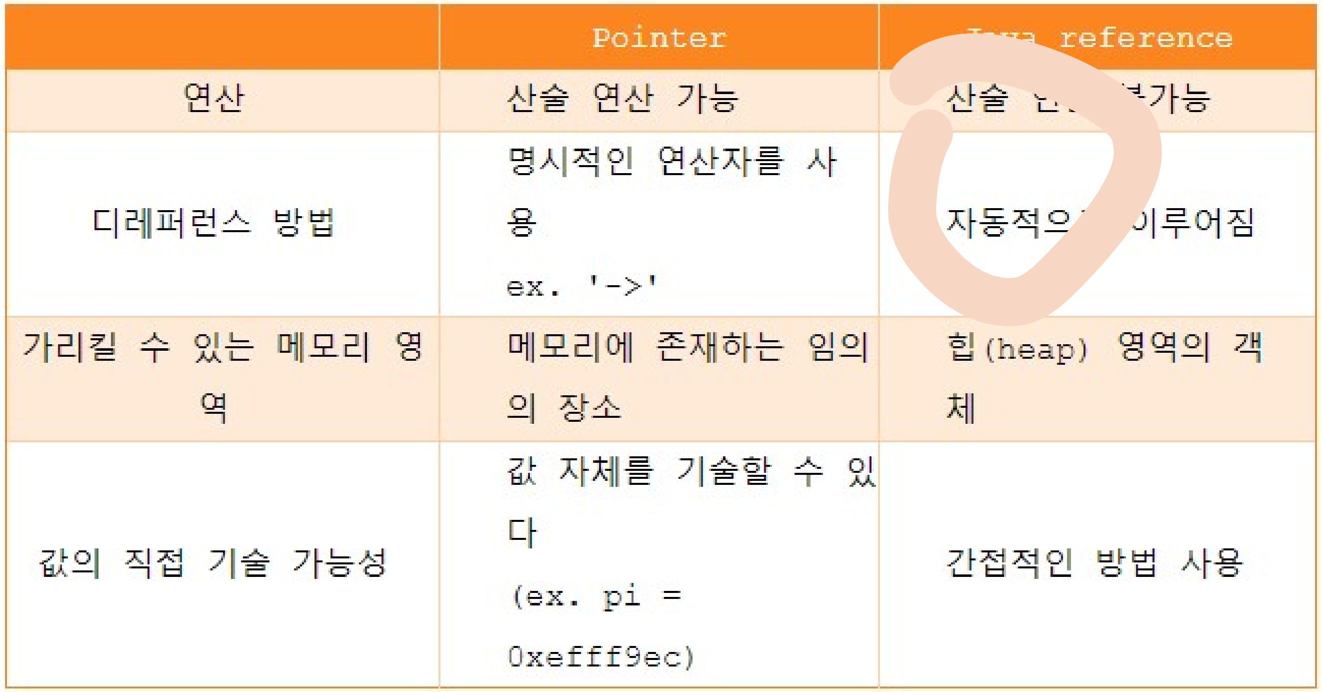
What does a variable hold?

» It depends on the type , *primitive* type or *class* type

- A primitive type variable holds [redacted] of the variable
- Class types
  - » have [redacted]s and instance [redacted]s
  - » holds the [redacted] of the object
  - » does not actually hold [redacted] of the object
  - » objects generally do not have a single value and they also have methods, so it does not make sense to talk about its "value"

포인터와 레퍼런스

- https://manorgass.tistory.com/25



	Pointer	Java reference
연산	산술 연산 가능	산술 연산 불가능
디레퍼런스 방법	명시적인 연산자를 사용 ex. '->'	자동적으로 이루어짐
가리킬 수 있는 메모리 영역	메모리에 존재하는 임의의 장소	힙(heap) 영역의 객체
값의 직접 기술 가능성	값 자체를 기술할 수 있다 (ex. pi = 0xefff9ec)	간접적인 방법 사용

Assignment with Variables of a Class Type

```
klingon.set("Klingon ox", 10, 15);
earth.set("Black rhino", 11, 2);
earth = klingon;
earth.set("Elephant", 100, 12);
System.out.println("earth:");
earth.writeOutput();
System.out.println("klingon:");
klingon.writeOutput();
```

What will the output be?

(see the next slide)

Assignment with Variables of a Class Type

```
klingon.set("Klingon ox", 10, 15);
earth.set("Black rhino", 11, 2);
earth = klingon;
earth.set("Elephant", 100, 12);
System.out.println("earth:");
earth.writeOutput();
System.out.println("klingon:");
klingon.writeOutput();
```

Output:

```
earth:
Name = Elephant
Population = 100
Growth Rate = 12%
klingon:
Name = Elephant
Population = 100
Growth Rate = 12%
```

What will the output be?

`klingon` and `earth` both print `Elephant`.

Why do they print the same thing?

(see the next slide)

Assignment with Variables of a Class Type

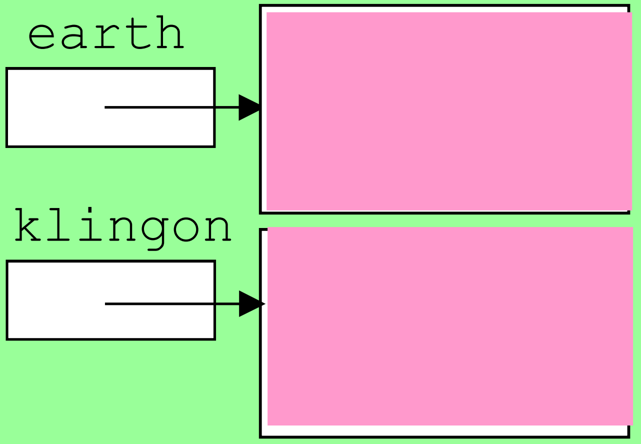
```
klingon.set("Klingon ox", 10, 15);
earth.set("Black rhino", 11, 2);
earth = klingon;
earth.set("Elephant", 100, 12);
System.out.println("earth:");
earth.writeOutput();
System.out.println("klingon:");
klingon.writeOutput();
```

Why do they print the same thing?

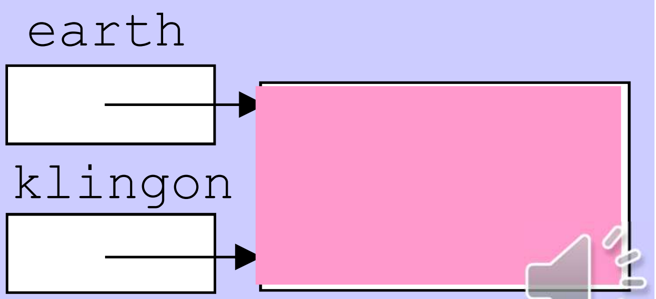
The assignment statement makes `earth` and `klingon` refer to the same object.

When `earth` is changed to "`Elephant`", `klingon` is changed also.(p. 233)

Before the assignment statement, earth and klingon refer to two different objects.



After the assignment statement, `earth` and `klingon` refer to the same object.



Assignment with Variables of a Class Type

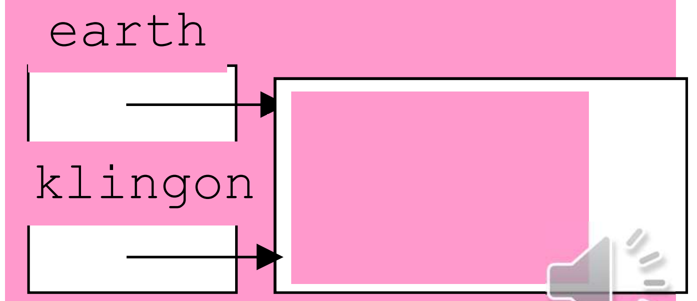
```
klingon.set("Klingon ox", 10, 15);
earth.set("Black rhino", 11, 2);
earth = klingon;
earth.set("Elephant", 100, 12);
System.out.println("earth:");
earth.writeOutput();
System.out.println("klingon:");
klingon.writeOutput();
```

Why do they print the same thing?

The assignment statement makes `earth` and `klingon` refer to the same object.

When `earth` is changed to "`Elephant`", `klingon` is changed also.(p. 233)

After the assignment statement, `earth` and `klingon` refer to the same object.



Gotcha: Comparing Class Variables

- A class variable returns a [redacted], but it is not its [redacted]
- It returns [redacted] where the object with that variable name is stored

If two class variables are compared u[illegible]ng ==,
it is the addresses, not the values that [illegible]pared!
This is rarely what you want to do!

- Use the class's `.equals()` method to compare the *values* of class variables

Example: Comparing Class Variables

```
//User enters first string
String firstLine = keyboard.nextLine();

//User enters second string
String secondLine = keyboard.nextLine();

if(firstLine == secondLine)
//this compares their addresses
{
   <body of if statement>
}

if(firstLine.equals(secondLine)
//this compares their values
{
   <body of if statement>
}
```

Use `.equals` method (not the double-equals sign) to compare values

Example: Comparing Class Variables

```
SpeciesFourthTry klingonSpecies = new SpeciesFourthTry();
SpeciesFourthTry earthSpecies = new SpeciesFourthTry();
klingonSpecies.set("klingon ox", 10, 15);
earthSpecies.set("Klingon ox",10,15);
If (klingonSpecies == earthSpecies)
    System.out.println("They are EQUAL.");
Else
    System.out.println("They are NOT equal.");


Output
```

[redacted] ✓

•

Example: Comparing Class Variables

```
SpeciesFourthTry klingonSpecies = new SpeciesFourthTry();
SpeciesFourthTry earthSpecies = new SpeciesFourthTry();
klingonSpecies = earthSpecies;
klingonSpecies.set("klingon ox", 10, 15);
earthSpecies.set("Klingon ox",10,15);
If (klingonSpecies == earthSpecies)
    System.out.println("They are EQUAL.");
Else
    System.out.println("They are NOT equal.");

Output
```

[redacted]

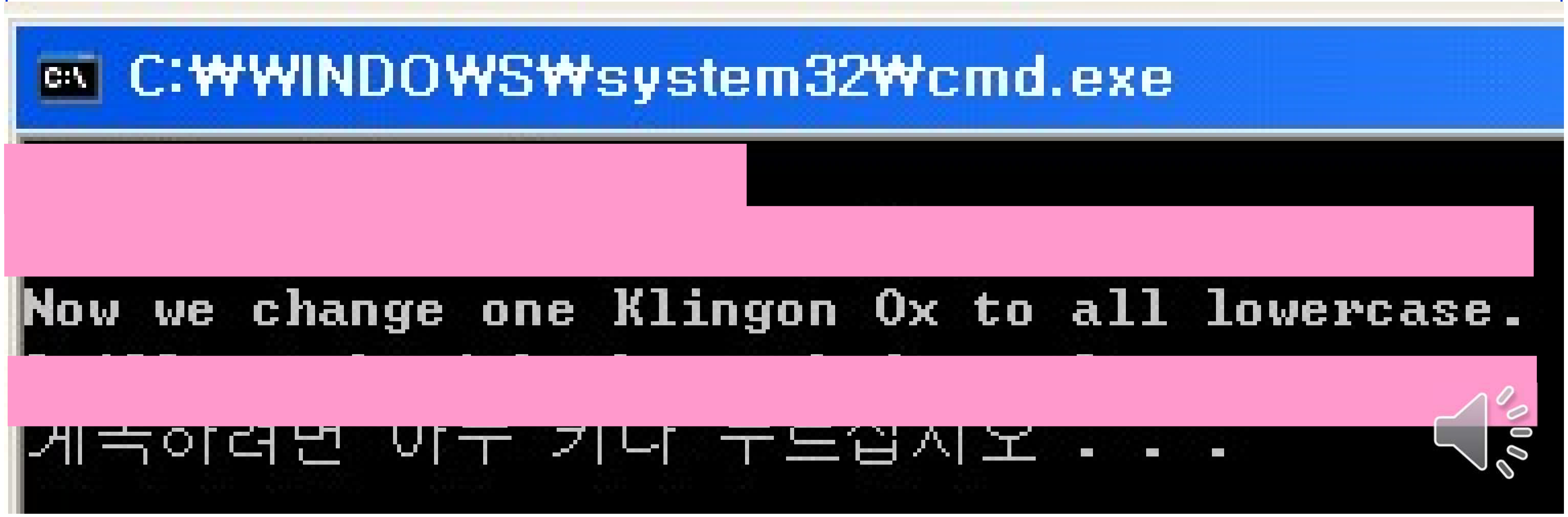
Listing 5.17 Defining an equals Method -Species.java

```
// Listing 5.17 (Defining an equals Method
/**
 Class for data on endangered species.
*/
public class Species
{
    private String name;
    private int population;
    private double growthRate;
    .................
    .................

     // equals
     public boolean equals(Species otherOb[illegible]t)
     {
        return (([redacted])
                && ([redacted]
                && ([redacted]
     }
}
```

LISTING 5.18 Demonstrating an Equals Method- SpeciesEqualsDemo.java

```
// LISTING 5.18 Demonstrating an equals Method

public class SpeciesEqualsDemo
{
   public static void main(String[] args)
   {
      Species s1 = new Species( ), s2 = new Species( );

      s1.set("Klingon Ox", 10, 15);
      s2.set("Klingon Ox", 10, 15); //uppercase K

      if (s1 == s2)
         System.out.println("Match with ==.");
      else
         System.out.println("Do Not match with ==.");
```

```
        if (s1.equals(s2))
            System.out.println("Match with the method equals.");
        else
            System.out.println("Do Not match with the method equals.");

        System.out.println(
                        "Now we change one Klingon Ox to all lowercase.");

        s2.set("klingon ox", 10, 15);  //lowercase k
        if (s1.equals(s2)) // equals : look at the Species class (equalsIgnoreCase)
            System.out.println("Still match with the method equals.");
        else
            System.out.println("Do Not match with the method equals.");
    }
}
```



C:\WINDOWS\system32\cmd.exe

```
[redacted]
Now we change one Klingon Ox to all lowercase.
[redacted]
계속하려면 아무 키나 누르십시오 . . .
```

Listing 5.19 Defining an equals Method -Species.java

```
// Lising 5.19
/**
 Class for data on endangered species.
*/
public class Species
{
    private String name;
    private int population;
    private double growthRate;
    ………………..
    ………………..

     // equals
     public boolean equals(Species otherObject)
     {
        return ((name.equalsIgnoreCase(otherObject.name))
                && (population == otherObject.population)
                && (growthRate == otherObject.growthRate));
     }
}
```

Pass the [redacted]: *Class* Types as Method Parameters

- In the same way, class variable names used as parameters in a method call copy the argument's [redacted] (not the value) to the formal parameter
- So the formal parameter name also contains the address of the argument
- It is as if the formal parameter name is an alias for the argument name

> Any action taken on the formal parameter is actually taken on the original argument!

- Unlike the situation with primitive types, the original argument is not protected for class types!

Example: Class Type as a Method Parameter

```
//Method definition with a DemoSpecies class
parameter
public void makeEqual(DemoSpecies otherObject)
{
   otherObject.name = this.name;
   otherObject.population = this.population;
   otherObject.growthRate = this.growthRate;
}

//Method invocation
DemoSpecies s1 = new DemoSpecies("Crepek", 10, 20);
DemoSpecies s2 = new DemoSpecies();
s1.makeEqual(s2);
```

- The method call makes `otherObject` an alias for `s2`, therefore *the method acts on `s2`, the `DemoSpecies` object passed to the method!*
- This is *unlike* primitive types, where the passed variable cannot be changed.

Listing 5.21 Class type parameters versus Primitive Type Parameters

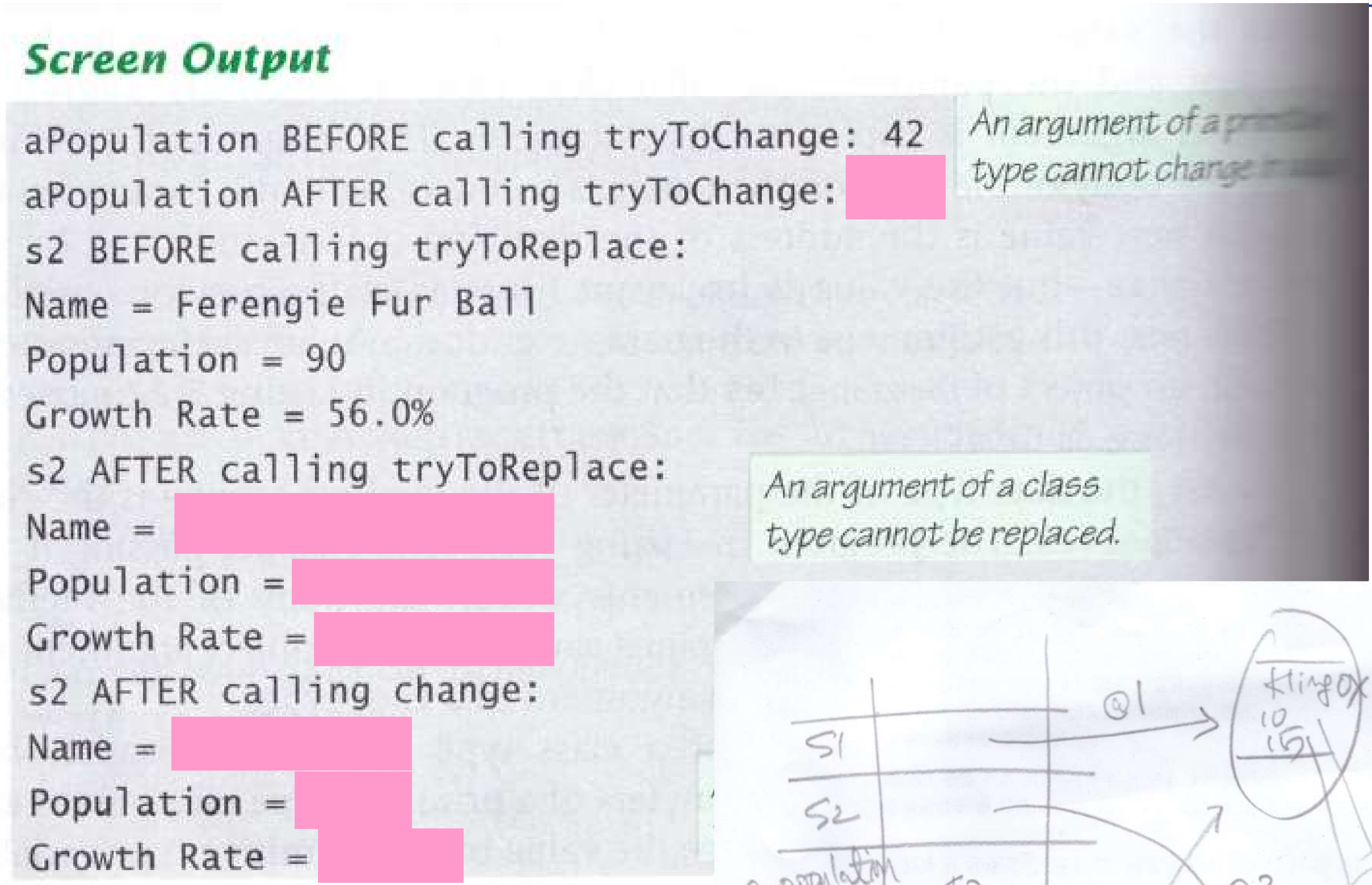
```
import java.util.Scanner;
/**
This version of the class Species is only a toy example designed
to demonstrate the difference between parameters of a class type
and parameters of a primitive type.
*/
public class DemoSpecies
{
   private String name;
   private int population;
   private double growthRate;

   /**
   Tries to set intVariable equal to the population of this object.
   But arguments of a primitive type cannot be changed.
   */
   public void tryToChange (int intVariable)
   {
      intVariable = this.population;
   }
```

```
/**
Tries to make otherObject reference this object.
But arguments of a class type cannot be replaced.
*/
public void tryToReplace (DemoSpecies otherObject)
{
    otherObject = this;
}


/**
Changes the data in otherObject to the data in this object,
which is unchanged.
*/
public void change (DemoSpecies otherObject)
{
    otherObject.name = this.name;
    otherObject.population = this.population;
    otherObject.growthRate = this.growthRate;
}

…………
```

Listing 5.22 Class type parameters versus Primitive Type Parameters

```
public class ParametersDemo
{
    public static void main (String [] args)
    {
        DemoSpecies s1 = new DemoSpecies (), s2 = new DemoSpecies ();
        s1.setSpecies ("Klingon ox", 10, 15);
        int aPopulation = 42;
        System.out.println ("aPopulation BEFORE calling tryToChange: "
                + aPopulation);  //@1
        s1.tryToChange (aPopulation);
        System.out.println ("aPopulation AFTER calling tryToChange: "
                + aPopulation);  //@2
        s2.setSpecies ("Ferengie Fur Ball", 90, 56);
        System.out.println ("s2 BEFORE calling tryToReplace: ");
        s2.writeOutput (); //@3
        s1.tryToReplace (s2);  //@4
        System.out.println ("s2 AFTER calling tryToReplace: ");
        s2.writeOutput ();
        s1.change (s2);  //@5
        System.out.println ("s2 AFTER calling change: ");
        s2.writeOutput ();
    } }
```

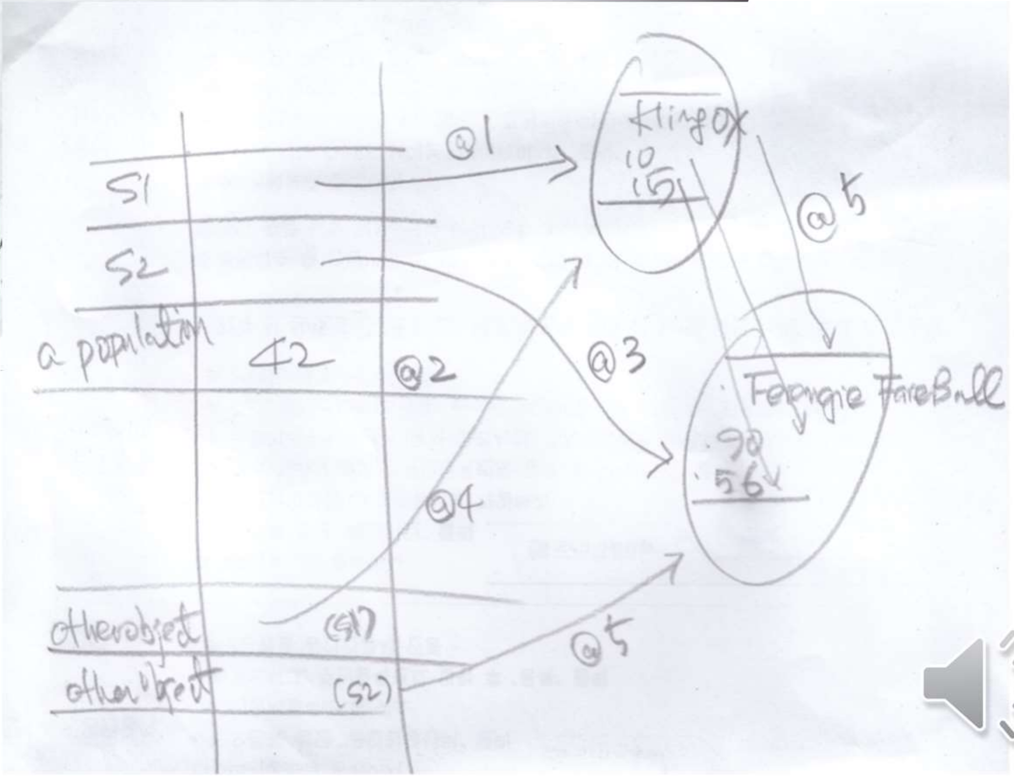
Screen Output



```
aPopulation BEFORE calling tryToChange: 42
aPopulation AFTER calling tryToChange: [redacted]
s2 BEFORE calling tryToReplace:
Name = Ferengie Fur Ball
Population = 90
Growth Rate = 56.0%
s2 AFTER calling tryToReplace:
Name = [redacted]
Population = [redacted]
Growth Rate = [redacted]
s2 AFTER calling change:
Name = [redacted]
Population = [redacted]
Growth Rate = [redacted]
```

An argument of a pri
type cannot change

An argument of a class type cannot be replaced.



끝