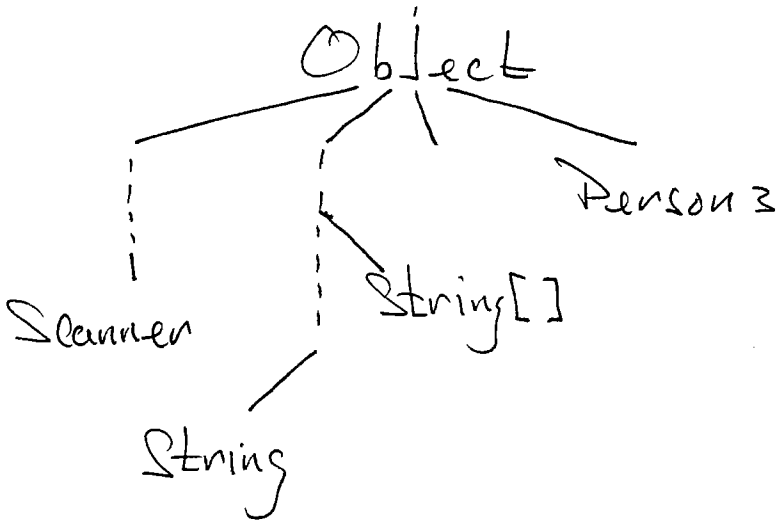
```
// Person4.java
// Includes constructors that perform initialization
class Person4{

   // Fields
   String name;
   String phoneNumber;
   int age;
   double weight;

   // Constructors
   Person4(String name){
      this.name = name;
   }

   Person4(String name, String phoneNumber){
      this.name = name;
      this.phoneNumber = phoneNumber;
   }

   Person4(String name, String phoneNumber, int age, double weight){
      this.name = name;
      this.phoneNumber = phoneNumber;
      this.age = age;
      this.weight = weight;
   }

   // Instance methods
   void printPerson4(){
      System.out.println("\nName: " + name);
      System.out.println("Phone Number: " + phoneNumber);
      System.out.println("Age: " + age);
      System.out.println("Weight: " + weight + "\n");
   }

   boolean isOlderThan(Person4 x){
      return (this.age>x.age);
   }
}

// Person4Program.java
// Uses the Person3 class
class Person4Program{
   public static void main(String[] args){
      Person4 a = new Person4("Dick", "123-456-7890", 6, 50.0);
      Person4 b = new Person4("Jane", "123-456-7890");

      // print out vital statistics
      a.printPerson4();
      b.printPerson4();

      // initialize the rest of b
      b.age = 5;
      b.weight = 40.0;

      // print out b again
      b.printPerson4();

      // compare ages
      if(a.isOlderThan(b))
         System.out.println(a.name + " is older than " + b.name);
      else if(b.isOlderThan(a))
         System.out.println(b.name + " is older than " + a.name);
      else
         System.out.println(a.name + " and " + b.name + " are the same age");
   }
}
```

RECALL THAT EVERY CLASS IN JAVA IS A SUBCLASS OF THE Object CLASS



A FULL DISCUSSION OF INHERITANCE IN JAVA MUST WAIT, BUT ONE FACT STANDS OUT: DESCENDENT CLASSES HAVE ALL MEMBERS (VARIABLES & methods) OF THEIR ANCESTOR CLASSES, AND POSSIBLY MORE.

i.e. IF CLASS A is AN ANCESTOR OF CLASS B, AND CLASS A HAS A method CALLED happy(), THEN CLASS B ALSO HAS A method CALLED happy().

i.e. B "INHERITS" ALL THE POSSESSIONS OF ITS ANCESTOR A. B MAY HAVE

MORE MEMBERS, BUT IT WILL HAVE AT LEAST WHAT A HAS.

SINCE Object IS A SUPERCLASS OF EVERY CLASS, ALL OF ITS methods ARE INHERITED BY EVERY JAVA CLASS.

Object HAS 11 Public methods AND 1 CONSTRUCTOR. SOME OF THESE WILL BE IMPORTANT TO US. IN PARTICULAR:

- toString()
- equals()
- clone()

THE toString() method IS USED TO CREATE A STRING REPRESENTATION OF AN OBJECT

AS DEFINED IN Object toString() returns THE MEMORY ADDRESS (IN THE JVM) OF AN INSTANCE OF A CLASS. YOU CAN OVERRIDE THIS DEFINITION HOWEVER TO CAUSE toString() TO RETURN WHATEVER String YOU LIKE.

114

```
// Person5.java
// Includes constructors that perform initialization
class Person5{

   // Fields
   String name;
   String phoneNumber;
   int age;
   double weight;

   // Constructors
   Person5(String name){
      this.name = name;
   }

   Person5(String name, String phoneNumber, int age, double weight){
      this.name = name;
      this.phoneNumber = phoneNumber;
      this.age = age;
      this.weight = weight;
   }

   // toString()
   // overrides Objects toString() method
   public String toString(){
      return (   "\nName: " + name + "\nPhone Number: " + phoneNumber
               + "\nAge: " + age + "\nWeight: " + weight  );
   }

   boolean isOlderThan(Person5 x){
      return (this.age>x.age);
   }
}

// Person5Program.java
// Uses the Person5 class
class Person5Program{
   public static void main(String[] args){
      Person5 a = new Person5("Dick", "123-456-7890", 6, 50.0);
      Person5 b = new Person5("Jane", "123-456-7890", 5, 40.0);

      // print out vital statistics
      System.out.println(a.toString());
      System.out.println(a);
      System.out.println(b);

      // compare ages
      if(a.isOlderThan(b))
         System.out.println(a.name + " is older than " + b.name);
      else if(b.isOlderThan(a))
         System.out.println(b.name + " is older than " + a.name);
      else
         System.out.println(a.name + " and " + b.name + " are the same age");
   }
}
```

NOTICE THAT THESE TWO CALLS PRODUCE THE SAME OUTPUT.

WE SEE THAT System.out.println(someObject) AUTOMATICALLY CALLS someObject's toString() METHOD, WHICH IS VERY CONVENIENT.

TRY COMMENTING OUT THE DEFINITION OF toString() IN Persons.java, RECOMPILE PersonsProgram.java, THEN SEE WHAT IS PRINTED.

I GET:

```
Persons@130c19b
Persons@130c19b
Persons@1f6a7b9
Dick is Older than Jane
```

THIS ILLUSTRATES THE DEFAULT BEHAVIOR OF Objects toString() method. WE SEE toString() is MEANT TO BE RE-DEFINED

NOTICE ALSO THE USE OF THE Public KEYWORD IN THE DEFINITION OF toString(). TRY LEAVING THIS OUT. THE COMPILER will OBJECT SAYING SOMETHING LIKE:

"ATTEMPT TO ASSIGN WEAKER ACCESS PRIVILEGES, WAS Public"

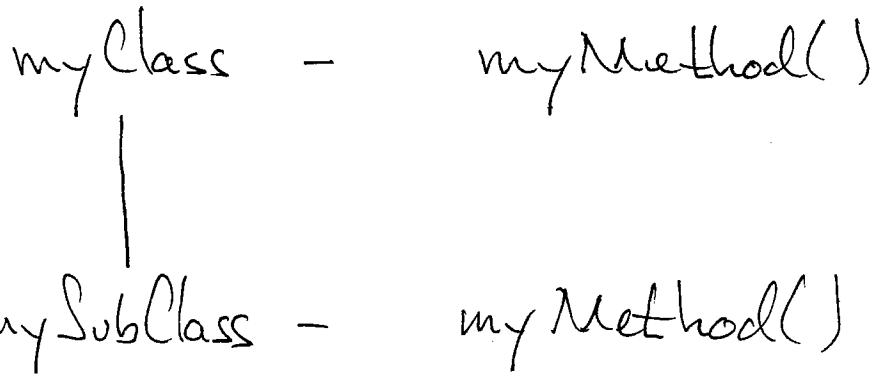
Object DEFINES toString() AS BEING Public, AND NO SUBCLASS CAN CHANGE THAT

JAVA HAS FOUR ACCESS LEVELS:

Private	—	NARROWEST
Package	—	DEFAULT
Protected		
Public	—	WIDEST

MORE ON ACCESS PRIVILEGES LATER.

IT IS IMPORTANT TO UNDERSTAND THE DISTINCTION BETWEEN OVERLOADING A METHOD AND OVERRIDING A Method.



TO OVERLOAD MEANS TO WRITE A NEW FUNCTION WITH THE SAME NAME (BUT DIFFERENT SIGNATURE). TO OVERRIDE MEANS TO REDEFINE ENTIRELY.

IN THE ABOVE PICTURE mySubClass INHERITS myMethod() from its SUPER CLASS myClass.

- TO HAVE mySubClass OVERLOAD myMethod(), GIVE A DEFINITION WITH A DIFFERENT SIGNATURE THAN APPEARS IN myClass.
- TO HAVE mySubClass OVERRIDE myMethod(), GIVE A DEFINITION WITH THE SAME SIGNATURE AS APPEARS IN myClass. IN DOING SO, YOU MAY NOT SET myMethod() TO HAVE A NARROWER ACCESS PRIVILEGE.

TO ILLUSTRATE THIS DIFFERENCE, RE-DEFINE Person'S toString() AS FOLLOWS:

```
String toString(int n){
    return( ..... );
}
```

SAME STRING

DIFFERENT ARGUMENT LIST, SO DIFFERENT SIGNATURE

NO Public ACCESS MODIFIER

THIS EXAMPLE OVERLOADS toString(), HENCE THERE IS NO NEED FOR THE Public ACCESS MODIFIER.

NOW IN PersonsProgram.java DO

```
System.out.println(a.toString(5));
// Prints: Dick
//         123-456-7890
//         ETC...
// AS BEFORE

System.out.println(a.toString())
// Prints: Persons@130c19b

System.out.println(a)
// Prints: Persons@130c19b.
```

ANOTHER METHOD FROM Object THAT IS MEANT TO BE OVERRIDDEN is equals(). ITS SIGNATURE IN Object is

```
Public boolean equals(Object obj)
```

ADD IN PersonsProgram.java

```
System.out.println(a.equals(b));
System.out.println(a.equals(a));
```

OUTPUT:

```
false
true.
```

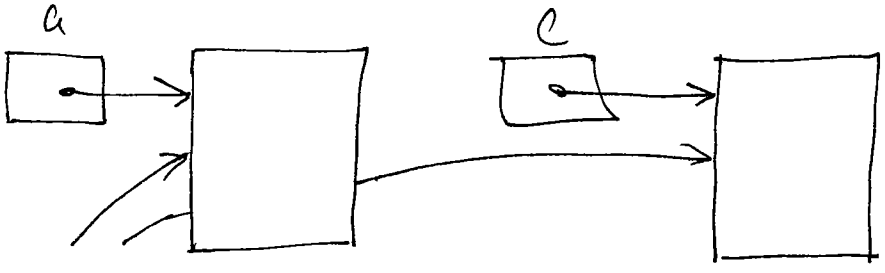
NOW CREATE A COPY OF a:

```
Persons c = new Persons("Dick", "123-456-7890",
                        6, 50.0);

System.out.println(a.equals(c));
```

OUTPUT:

```
false
```

BY DEFAULT equals() JUST COMPARES THE ADDRESSES (IN TURN) OF THE TWO OBJECTS. THUS THE PICTURE

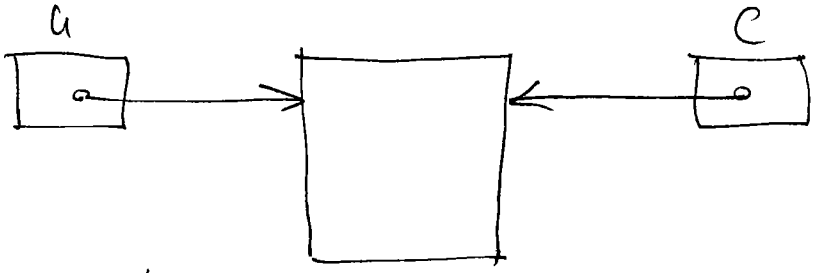


SAME DATA BUT DIFFERENT ADDRESSES.

CAUSES a.equals(c) TO RETURN false. IF INSTEAD WE DID

Persons c = a;

WE HAVE THE PICTURE



a & c POINT TO SAME MEMORY

IN THIS CASE a.equals(c) RETURNS TRUE.

THIS IS THE DEFAULT BEHAVIOR OF equals(). OFTEN WE WISH TO OVERRIDE equals() SO AS TO COMPARE THE ACTUAL DATA, NOT THE ADDRESSES.

TO DO THIS WE DO FROM WITHIN Persons.java

```
Public boolean equals(Object x) {
    Persons p = (Persons) x;
    return( this.name.equals(P.name)
            && this.phoneNumber.equals(P.phoneNumber)
            && this.age == P.age
            && this.weight == P.weight );
}
```

Now back in PersonsProgram.java

```
Persons e = new Persons( p );
```

SAME DATA AS a

```
System.out.println( a.equals(e) );
```

OUTPUT:
true

NOTICE THAT THE SIGNATURE OF equals() above is IDENTICAL TO THAT ONE IN Object. IN PARTICULAR ITS PARAMETER is Object, NOT Persons. IF WE USED Persons INSTEAD, WE WOULD BE OVERLOADING NOT OVERIDING.

NOTICE ALSO THE LOCAL VARIABLE

```
Persons p = (Persons) x;
```

CAST OBJECT AS PERSONS

THIS IS NECESSARY SINCE THE COMPILER WOULD OBJECT TO AN EXPRESSION LIKE

```
x.name
```

SINCE Object DOES NOT HAVE A name field..

ALSO NOTICE THAT WHEN WE COMPARE Strings WE USE equals() WHILE IF WE COMPARE ints OR doubles WE USE ==.

THIS IS AGAIN BECAUSE OF THE NATURE OF REFERENCE TYPES VS. PRIMITIVE TYPES.