CHAPTER 6
OBJECTS AND CLASSES

A class in Java is a new data type that bundles one or more data values, together with some methods that operate on that new data type.

The first thing to realize is that a class need not contain anything. In fact

```
class empty {
}
```

is a valid Java class. In particular function main() is not required.

Of course if there is no main() there is no program entry point, and therefore no program.

What may a class contain?

- variables called member variables, or fields.
- methods called member methods.
- other classes.

TYPICALLY WE USE A CLASS TO MODEL OR SIMULATE SOME CONCEPT.

THE EXAMPLE Person.java IS A DATA TYPE MEANT TO REPRESENT A PERSON.

```
class Person {
    String name;
    String phoneNumber;
    int age;
    double weight;
}
```

NOTICE THIS CLASS HAS NO MAIN() METHOD AND HENCE IS NOT A PROGRAM. IT IS INSTEAD A NEW DATA TYPE.

class Person CONTAINS FOUR MEMBER VARIABLES AND NO methods. THE DEFINITIONS OF THE MEMBER VARIABLES MAY LOOK LIKE VARIABLE DEFINITIONS, BUT THEY ARE NOT. NO MEMORY IS ALLOCATED BY THESE STATEMENTS.

100

```
// Person.java
// A very simple simulation of a person.  Notice there is no main() method,
// no methods at all in fact, so this is not a program.  It is instead a new
// data type.
class Person{
   String name;
   String phoneNumber;
   int age;
   double weight;
}
```

```
// PersonProgram.java
// Uses the Person class
class PersonProgram{
   public static void main(String[] args){
      Person a;
      Person b;

      a = new Person();
      b = new Person();

      a.name = "Dick";
      a.phoneNumber = "123-456-7890";
      a.age = 6;
      a.weight = 50.0;

      b.name = "Jane";
      b.phoneNumber = "123-456-7890";
      b.age = 5;
      b.weight = 40.0;

      System.out.println(a.name);
      System.out.println(b.phoneNumber);
   }
}
```

PersonProgram.java is a class with a main() method, and is hence a program. This program uses the Person data type.

The statement

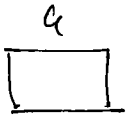
a = new Person();

allocates memory for a new Person object, i.e. an instance of the

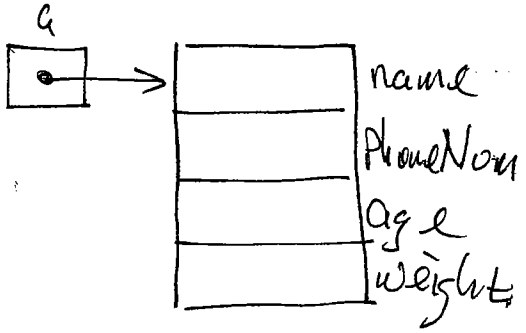
PERSON CLASS, AND ASSIGNS IT TO a, A VARIABLE OF TYPE Person.

NOTE THAT ALL CLASS VARIABLES ARE REFERENCE VARIABLES, AND HENCE STORE THE ADDRESS (IN THE JVM) OF AN INSTANCE OF THAT CLASS.

AFTER Person a;

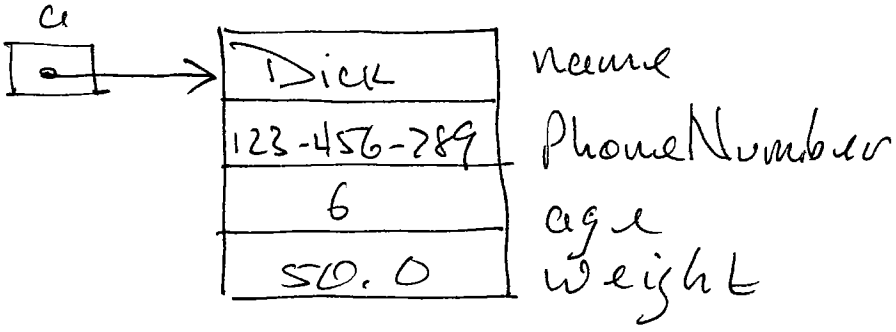


AFTER a = new Person();



NOTICE THAT A Person OBJECT IS A LOT LIKE AN ARRAY OBJECT, EXCEPT THAT NOT ALL ELEMENTS OF A Person HAVE THE SAME TYPE.

AFTER INITIALIZATION:



IT'S NOT QUITE RIGHT TO SAY THAT THE Person CLASS HAS NO METHODS. EVERY CLASS HAS A constructor method WHOSE NAME IS THE SAME AS THE CLASS, IN THIS CASE

Person()

IS THE Person constructor. IT IS CALLED BY THE new OPERATOR AND BY DEFAULT, IT JUST TELLS new HOW MUCH MEMORY TO ALLOCATE.

WE CAN EXPLICITLY DEFINE THE CONSTRUCTOR TO PERFORM OTHER TASKS, LIKE INITIALIZATION. MORE ON THIS LATER.

THE DOT "." IN EXPRESSIONS LIKE

a.age
b.weight

IS KNOWN AS THE MEMBER ACCESS OPERATOR. IT IS VERY

MUCH LIKE THE INDEX BRACKETS "[ ]" IN AN ARRAY VARIABLE

list[6]

IN THAT IT ALLOWS US TO ACCESS THE INDIVIDUAL ELEMENTS OF A PERSON OBJECT.

THIS SIMPLE CLASS IS VERY SIMILAR TO A struct IN C OR C++, I.E. A WAY FOR PROGRAMMER TO AGGREGATE RELATED DATA FIELDS INTO A SINGLE OBJECT.

THE BEAUTY OF THE CLASS CONSTRUCT HOWEVER, IS THAT IT ALLOWS US TO ENCAPSULATE NOT JUST DATA FIELDS, BUT THE methods WHICH MOST NATURALLY OPERATE ON THOSE FIELDS.

FOR INSTANCE, WE MIGHT WANT TO PRINT OUT ALL THE VITAL STATISTICS OF A Person OBJECT.

ONE WAY TO PROCEED WOULD BE TO WRITE A STATIC FUNCTION IN PersonProgram.java TO DO THIS:

```
class PersonProgram {
    Public static void main(String[] args){
        :
        :
        PrintPerson(a);
        PrintPerson(b);
    }

    static void PrintPerson(Person x){
        System.out.println("Name:" + x.name);
        System.out.println("Phone:" + x.PhoneNumber);
        System.out.println("Age:" + x.age);
        System.out.println("Weight:" + x.weight);
    }

}
```

OUTPUT:

```
Name: Dick
Phone: 123-456-789
Age: 6
Weight: 50.0
  :
  :
```

However the PrintPerson operation is more closely associated with the Person class itself. We may therefore choose to include it in the Person class as a member method.

PrintPerson can be included in class Person in two ways. One would be to make it a static method, just as above. It would then be called from PersonProgram.java by doing

```
Person.PrintPerson(a);
```

In general, static methods are accessed from outside their class by

```
ClassName.MethodName(...args...)
```

↑

Member access operator.

106

```
// Person2.java
// Includes a static method to print out a Person object
class Person2{
   String name;
   String phoneNumber;
   int age;
   double weight;

   static void printPerson(Person2 x){
      System.out.println("Name: " + x.name);
      System.out.println("Phone Number: " + x.phoneNumber);
      System.out.println("Age: " + x.age);
      System.out.println("Weight: " + x.weight);
   }
}


// Person2Program.java
// Uses the Person2 class
class Person2Program{
   public static void main(String[] args){
      Person2 a;
      Person2 b;

      a = new Person2();
      b = new Person2();

      a.name = "Dick";
      a.phoneNumber = "123-456-7890";
      a.age = 6;
      a.weight = 50.0;

      b.name = "Jane";
      b.phoneNumber = "123-456-7890";
      b.age = 5;
      b.weight = 40.0;

      System.out.println(a.name);
      System.out.println(b.phoneNumber);

      System.out.println();
      Person2.printPerson(a);
      System.out.println();
      Person2.printPerson(b);
      System.out.println();
   }
}
```

NOTICE HOW printPerson() IS NOW CALLED.

THIS EXAMPLE BUNDLES THE PrintPerson() METHOD IN THE CLASS Person2. ITS OUTPUT IS IDENTICAL TO THE PREVIOUS PROGRAM.

107

```
// Person3.java
// Includes instance methods to print out a Person3 object, and to compare
// the ages of different Person3 objects.
class Person3{
   String name;
   String phoneNumber;
   int age;
   double weight;

   void printPerson3(){
      System.out.println("Name: " + name);
      System.out.println("Phone Number: " + phoneNumber);
      System.out.println("Age: " + age);
      System.out.println("Weight: " + weight);
   }

   boolean isOlderThan(Person3 x){
      return (this.age>x.age);
   }
}



// Person3Program.java
// Uses the Person3 class
class Person3Program{
   public static void main(String[] args){
      Person3 a = new Person3();
      Person3 b = new Person3();

      // initialize a
      a.name = "Dick";
      a.phoneNumber = "123-456-7890";
      a.age = 6;
      a.weight = 50.0;

      // initialize b
      b.name = "Jane";
      b.phoneNumber = "123-456-7890";
      b.age = 5;
      b.weight = 40.0;

      // print out stats
      System.out.println();
      a.printPerson3();
      System.out.println();
      b.printPerson3();
      System.out.println();

      // 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");
   }
}
```

THIS SESSION INCLUDES INSTANCE METHODS TO PRINT OUT A PERSON3 OBJECT AND TO COMPARE AGES OF PERSONS OBJECTS.

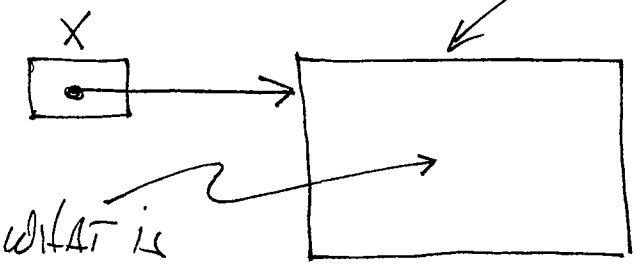
Every class in Java has a constructor method whose name is the name of the class itself. Constructors are special methods that are only called in conjunction with the new operator. Basically, the constructor tells new how much memory to allocate, and possibly how to initialize the object.

The general declaration + allocation statement looks like

```
myClass x = new myClass();
```

- myClass: class name
- x: variable of type myClass (Necessarily a reference variable)
- myClass(): the constructor for the myClass class

Result:



The details of what is inside are specified by myClass Definition.

```
class myClass {
    // fields i.e. member variables
    // methods
}
```

IT IS POSSIBLE FOR THE PROGRAMMER TO ADD ONE (OR MORE) CONSTRUCTORS TO THE DEFINITION TO PERFORM INITIALIZATION TASKS.

EX

```
class myClass {
    int happy;
    int sad;

    myClass(int a, int b) {
        happy = a;
        happy = b;
    }

    int getHappy() {
        return happy;
    }
}
```

NOTICE THAT ONE DOES NOT SPECIFY A RETURN TYPE FOR A CONSTRUCTOR

Now in some method, in some (possibly other) class

```
myClass x = new myClass(7, -12);

System.out.println( x.getHappy() );
System.out.println( x.happy );
System.out.println( x.sad );

// Prints 7, 7, -12 respectively.
```

Notice that it is somewhat unnatural to have formal parameters a and b in the myClass constructor. happy and sad would make more sense, but these are the field names. This is remedied by the "this" reference

```
myClass(int happy, int sad){
    this.happy = happy;
    this.sad = sad;
}
```

The Person4 class has several constructors.