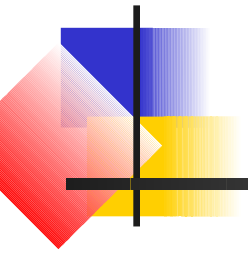
Programming with Java

typing, classes and objects

P. Bakowski

bako@ieee.org



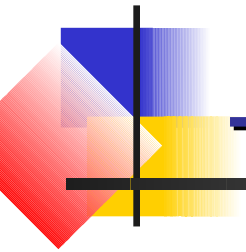
The types – three kinds

There are three kinds of types that can be used to declare a variable, a parameter, or a value calculated and returned by a method:

1. primitive types or base types :
   `int`, `double`, `boolean` and `char`.

2. object types – defined by a class

3. array types
type `T[]` : `T` – primitive type, object type, array type



Typing – verification and compilation

The compiler has to verify that :

- the value stored in a variable in an assignment is the same type as those reported for the variable

- an operator is applied to the values of the right type

- a method is called with the correct number of parameters and that these parameters have the type declared in the method definition

Conversions between primitive types

```
public class ExConv{
public static void main(String[] args) {
double d ;
int i ;
d = 10;
d = 10.3 + 10;     // int → double
System.out.println(d);
i = 'a';
i = 5 + 'a';       // char → int
System.out.println(i);}
}
```

```
Java/Test> java ExConv
20.3
102
```

Explicit conversions

```
public class ExConv3{
  public static void main(String[] args) {
  int i ;
  i = (int)10.7;
  System.out.println(i);
  }
}

public class ExConv4{
  public static void main(String[] args) {
  double d ;
  d = (double) 555;
  System.out.println(d);
  }
}
```

Conversions by method call

```
public class ExConv3{
  public static void main(String[] args) {
  int i ;  String s = « 1234 » ;
  i = (int) s;  // erreur !
  System.out.println(i);
  }
}
```

```
public class ExConv3{
  public static void main(String[] args) {
  int i ;
  String s = «1234» ;  // s is an object
  i = Integer.parseInt(s) ;
  System.out.println(i);
  }
}
```

Primitive types and classes

Each primitive type is associated with a predefined class that contains methods for conversions in both directions to other types.
Thus the `int` type is associated with the `class Integer` that offers various conversion methods:

- `String` to `int` : `Integer.parseInt()`
- `int` to `String` : `Integer.toString()`

- ..

type	class
`int`	`Integer`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

char and Character

Methodes to test :-

```
Character.isDigit()
Character.isLetter()
Character.isLowerCase()
Character.isUpperCase()
```

Methodes for conversion :

```
Character.toLowerCase()
Character.toUpperCase()
Character.toString()
```

```
public class ConvChar{
  public static void main(String[] args) {
  char c ;
  for(int i=32;i<300;i++){
  c = (char) i;
  System.out.println("the character"+c+"is");
  if(Character.isDigit(c)){
  System.out.println("a number");}
  else if ..
```

The control of decimal places

```
public class TDouble{
  public static void main(String[] args) {
  double d = 956.3582737839661;
  double d2 = ((int)(d*100.0)/100.0);
  double d3 = Math.round(d*100.0)/100.0;
  System.out.println("d="+d+"  d2="+d2+"  d3="+d3);
  }
}
```

For more than two decimal places, we can multiply a number by 100.0, take the integer part by converting it to an **int**, then divide by 100.0.

The classes and the Objects

A class is a prototype that defines the variables and methods common to all objects of the same kind.

A class is a template for objects.

Each class defines the way to create and manipulate objects of this type.

Conversely, an object is always a copy, an instance of a class.

Thus, to do the object programming, you must know how to design classes, to define object models, and create objects from these classes.

The classes and the objects

The design of a class is defined by:
1. Characteristic data objects of the class. We call these data instance variables.
2. The actions that can be performed on objects of the class. These are the methods that can be invoked on objects of the given class.

Thus, each created object possess:
1. State, it is specific values for the instance variables of the class to which it belongs.
2. Methods that will act on this state.

Class Account

```
class Account{
int balance ;
String titulaire;
int number ;
void display( ){
System.out.println("balance: "+this.balance);
}
void deposit(int amount){
balance = balance + amount ;
}
void withdraw(int amount){
balance = balance - amount ;
}
}
```

Attention !

An object method does not include the **static** keyword.

Using a class (`Account`)

```
public class TestAccount{
  public static void main(String[] args) {
  Account c1 = new Account();
  System.out.println(c1.balance);
  }
}
```

After executing `c1 = new Account ();` each instance variable `c1` has a default value. This value is `0` for balance and number, and `null` for the holder.

Access to instance variables

```
public class TestAccount1{
  public static void main(String[] args) {
  Account c1 = new Account ( ) ;
  int x ;
  int []tab = {2,4,6};
  tab[c1.balance]=3;
  tab[1]=c1.number ;
  x = d1.balance+34/(d1.number+4);
  }
}
```

Modification of instance variables

```
public class TestAccount2{
  public static void main(String[] args) {
  Account c1 = new Account();
  Account c2 = new Account();
  c1.balance = 100;  c1.number = 218;
  c1.holder ="Dupont";
  c2.balance = 200;  c2.number = 111;
  c2.holder ="Durand";
  System.out.println
  ("value of c1:"+c1.balance+","+c1.holder+","+c1.number);
  System.out.println
  ("value of c2:"+c2.balance+","+c2.holder+","+c2.number);
  }
}
```

Modification of instance variables

```
public class TestAccount2Bis{
  public static void main(String[] args) {
  Account c1 = new Account();
  Account c2 = new Account();  // no need of new Account() !
  c1.balance = 100;
  c1.number = 218;
  c1.holder ="Dupont";
  c2 = c1;
  c2.balance = 60;
  System.out.println
  ("value of c1:"+c1.balance+","+c1.holder+","+c1.number);
  System.out.println
  ("value of c2:"+c2.balance+","+c2.holder+","+c2.number);
  }
}
```

```
Java/Test> java TestAccount2Bis
value of c1: 60 , Dupont, 218
value of c2: 60 , Dupont, 218
```

Calling the methods on objects

```
public class TestAccountDisplay{
  public static void main(String[] args) {
  Account c1 = new Account();
  Account c2 = new Account();  // no need of new Account()
  c1.balance = 100;
  c1.number = 218;
  c1.holder =”Dupont”;
  c1.display();
  c2.display();
  }
}
```

```
Java/Test> java TestAccountDisplay
balance: 100
balance: 0
```

Keyword this

The **this** keyword refers to the object on which the method will be invoked.

For example, in view of the definition:

```
void display( ){
System.out.println("balance: "+this.balance);
}
```

this.balance denotes the value of the instance variable balance the object on which the method is invoked.

During execution of c1.**display**(), c1 designate **this**, while during execution of c2.**display**(), c2 designate **this**.

Invocation of methods with arguments

```
public class TestDisplay2{
  public static void main(String[] args) {
  Account c1 = new Account();
  c1.balance = 100;
  c1.number = 218;
  c1.holder =”Dupont”;
  c1.display();
  c1.deposit(800);
  c1.display();
  }
}
```

The `deposit` method modifies the state of the current object. In our example using the first `c1.display()` displays 100, while the second `c1.display()` displays 900.
Between these two actions, the execution of `c1.deposit(800)` changed the state of `c1`.

The methods with return value

```
class Account {
int balance ;
String holder;
int number ;
void display( ){
  System.out.println("balance: "+this.balance);}
int deposit(int amount){
  this.balance = this.balance + amount;
  return this.balance;}
}
```

```
public class TestDepose {
  public static void main(String[] args){
  Account c1 = new Account();
  c1.balance = 100;
  c1.number = 218;
  c1.holder ="Dupont";
  System.out.println(c1.deposit(800));}
}
```

Instance variables

```
class MyDate {
int day ;
int month;
int year ;
public void showDate( ){
  System.out.println
  (this.day+","+this.month+","+this.year);
  }
}
```

```
class Person {
  MyDate birth;
  String name ;
 // no methods to show
}
```

Instance variables

```
public class PersonTest {
  public static void main(String[] args){
  Person p2 = new Person();
  p2.name ="toto";
  p2.birth = new MyDate( );
  p2.birth.day = 18;
  System.out.println(p2.birth.day);
  System.out.println(p2.name + "date birth:");
  p2.birth.showDate();
  }
}
```

```
class Person {
  MyDate birth;
  String name;
 // no methods to show
}
```

Classes with `static` methods

```
class MyDate2 {
int day;int month;int year;
public void showDate(){
  System.out.println
  (this.day+","+this.month+","+this.year);}
public static boolean bissextile(int a){
return((a%4==0)&&(!(a%100==0)||a%400==0));}
public static int length(int m,int a){
if(m==1||m==3||m==5||m==7||m==8||m==10||m==12)
{ return 31;}
else if (m==2) {if(bissextile(a)){return 29;} else
{ return 28;}}
else {return 30;}
}
```

Declaring a static method means that this method does not know, in any way, the current object.

In other words, it is a method that acts only on its arguments and its return value.

Static methods are called with a class name: `Date.length(12.2000);`

Classes with `static` methods

```
class MyDate3 {
int day;int month;int year;
public void showDate(){ Keyboard.writeStringln
  (this.day+","+this.month+","+this.year);}
public static boolean bissextile ..

public static int length..

public void gotoTomorrow(){
if(this.day<length(this.month,this.year)){
this.day = this.day +1;}
else if(this.month==12){
this.day=1;this.year=this.year+1;this.month=1;}
else { this.day=1;this.month=this.month+1;}
}
```

Classes with static methods

```
public class DateTest3 {
  public static void main(String[] args){
MyDate3 d2 = new MyDate3();
d2.year = 2000;
d2.day = 28 ;
d2.month = 2 ;
d2.showDate();
d2.gotoTomorrow();
d2.showDate();
}
}
```

```
Java/Test> java DateTest3
28 , 2 , 2000
29 , 2 , 2000
```

The constructor by default

What happens when we write:

```
Date d1 = new Date(); ?
```

The `new` operator reserves the memory space required for the object d1 and initializes the data with default values. An instance variable of type `int` will receive `0`, a `boolean false` , a `char '\0'` and instance variables of an object type will be `null`.

The `Date` class contains no explicit definition of constructor.
And yet the created object can be referenced after the `new` operator.
This means that Java provides for each defined class a default constructor.
This default constructor:

- has the same name as the class and
- has no argument.

```
public Date(){   }
```

The constructors

A constructor is defined as a method except that:
1. The name of a constructor is always the class name.
2. A constructor never returns type.

In a class, you can define as many constructors as you want, as long as they differ in their number (or type) of arguments.

```
public class MyDate4 {
int day ; int month ; int year ;

public MyDate4( ) {
this.day = 1; this.month = 1 ; this.year = 1;
}

public MyDate4(int j, int m, int a ) {
this.day = j; this.month = m ; this.year = a;
}
// ....
}
```

The constructors

```
public class DateTest4 {
  public static void main(String[] args){
  MyDate4 d1 = new MyDate4();
  MyDate4 d2 = new MyDate4(1,12,2000);
  d1.showDate();
  d2.showDate();
  d2.gotoTomorrow();
  d2.showDate();
  }
}
```

```
Java/Test> java DateTest4
1 , 1 , 1
1 , 12 , 2000
2 , 12 , 2000
```

Data protection

To ensure the protection of data, there is the possibility to declare variables as private ; a private variable can be used only within the class. This is done through the keyword **private**.

```
class Account {
private int balance ; private String holder ;
private int number ;
Account(int num, String tit ) {   // constructor
holder = tit; number = num ; balance = 0; }
int getNumber(){ return number ;}
String getHolder(){ return holder ; }
int getBalance(){ return balance ;}

void display(){
  System.out.println("balance"+ this.balance);}
void deposit(int amount){
  this.balance = this.balance + amount;}
void withhold(int amount ){
  this.balance = this.balance - amount;}
}
```

Data protection

```
public class Other {
  public static void main(String[] args){
  Account c1 = new Account(100010,”Christian”);
  c1.balance = 100;
  c1.number= 1;
  c1.holder = ”Charles”; }
}
```

```
Java/Test> javac Autre.java
Autre.java:4: balance has private access in Compte
c1.solde = 100;
^
Autre.java:5: number has private access in Compte
c1.numero = 1;
^
Autre.java:6: holder has private access in Compte
c1.titulaire = "Charles";
^
3 errors
```

Static variables

We already know that the classes can contain:
1. instance variables
2. constructors
3. static or non-static methods.

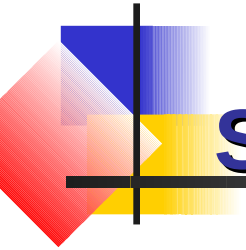
Classes may contain another kind of elements: static variables.

```
public class MySDate {
// instance variables
int day;
int month;
int year;
// static variable – class variable
static int nb ;
public MySDate (int d, int m, int y){
  day=d ; month=m ; year=y ;
  }
}
```

The behaviour of static variables

```
public class TestStaticDate {
  public static void main(String[] args){
  MySDate d1= new MySDate(1,1,2000);MySDate d2= new MySDate(2,2,2004);
  System.out.println(MySDate.nb); // access nb by MySDate
  System.out.println(d1.nb); // access nb by d1
  System.out.println(d2.nb); // access nb by d2
  MySDate.nb = 2; // static variable assignment
  System.out.println(MySDate.nb+","+d1.nb+","+d2.nb);
  d1.nb = 1; d1.day = 23;
  System.out.println(MySDate.nb+","+d1.nb+","+d2.nb);
  System.out.println(d1.day+","+d2.day); }
}
```

```
Java/Test> java TestStaticDate
0
0
0
2, 2 , 2
1, 1 , 1
23 , 2
```



Summary

Types, three kinds: primitives, objects, arrays

Conversion of primitive types

`char` and `Character`, `Integer` and `int` ..

Classes and Objects

Objects and instance variables

- methods
- constructors

Privacy

Static variable (class variable)