1.3. Computer memory, variables and objects

Computer memory. On binary computers as we use them, information is encoded into a sequence of so called bits (binary digits). A bit is either on (1) or off (0). Computer memory can be thought of as a long strip of cells containing 0s or 1s. Eight bits are collected in a so called byte and each byte has a memory address as shown in Figure 1.7.

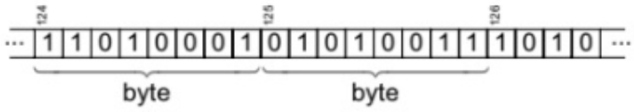


Figure 1.7: Computer memory

Variables of different data types. A variable can be thought of as a place in computer memory where your program can store information. Figure 1.8 shows two variables **a** and **b** that both occupy some amount of memory.

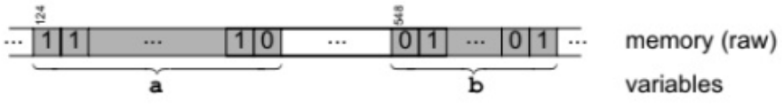


Figure 1.8: Variables

Variable – mjesto u memoriji kompjutera gdje program pohranjuje vrijednosti

In Java, every variable has a data type that specifies how the sequence of bits is decoded. Java distinguishes between two different types of variables: **primitive type variables** and **object type variables**. Both types have a specific meaning.

→ data type određuje kako se sekvenca bitova dekodira

Primitive type variables. In Java, there are eight basic data types like `int` and a basic data type representing "nothing", called `void`. A complete listing of primitive types is given in Table 1.1. For primitive type variables, we can say: → sadrži primitivnog tipa variable je VALUE

The content of a primitive type variable is the value.

Using the type information, the bit sequence for one variable can be decoded into a human readable format. Figure 1.9 shows a piece of Java code and the corresponding memory state. From now on, we will only use the decoded represention of computer memory.

Because primitive type variables store the value, assigning a value to a primitive type variable changes the content of the variable's memory. Figure 1.10 shows a code sequence and the corresponding memory state.

→ S inf. koji je tip variable u pitanju sekvenca bitova se pretvara u formu koju covjek razumije

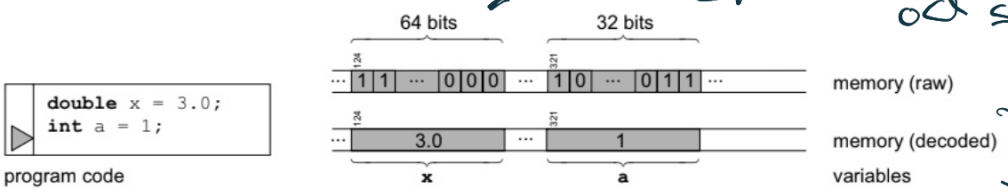
→ "=" - assigning operator ⇒ dodjeljuje vrijednost primitivnoj varijabli, tj. mijenja sadrzaj memorije variable

Table 1.1: The primitive data types of Java*

Type	Contains	Size	Range
`byte`	integer	8 bits	-128 ... 127
`short`	integer	16 bits	-32768 ... 32767
`int`	integer	32 bits	-2147483648 ... 2147483647
`long`	integer	64 bits	$-2^{63} \ldots 2^{63}-1$
`float`	floating-point	32 bits	ca. $\pm 3.40282347 \cdot 10^{38}$
`double`	floating-point	64 bits	ca. $\pm 1.79769313486231570 \cdot 10^{308}$
`boolean`	boolean value	1 bit	`true` or `false`
`char`	unicode character	16 bits	ISO Unicode character set
`void`	nothing	–	–

*The three most important data types are shaded grey.

svaka varijabla zauzima određen broj bit-ova (64 ili 32 zavisno od sistema)



zavisno od tipa, ta sekvenca 0 i 1 kodira neku vrijednost koju smo razumjeli

Figure 1.9: Primitive type variables

deklarisemo varijablu, dodijelimo vrijednost

"assigning value" - mijenjamo sadržaj u memoriji

Figure 1.10: Assigning to a primitive type variable

Object type variables. An object can be thought of as a collection of data that resides somewhere in memory. For object type variables, we can say:

object-type varijabla

The content of an object type variable is the address of an object.

→ sadržaj je objekt

→ object-type variable ne sadrže vrijednost tog objekta, već sadrže njihovu adresu

When an object is created via the operator **new**, the address is returned and can be stored in a variable. When we assign a new address to an object type variable, the variable value changes to the new address. This is illustrated in Figure 1.11 (we use the #-sign to indicate that a number represents an address). First, we have two variables both pointing to two distinct boxes. After the assignment, both variables point to one box and none to the other.

→ Kreiranjem objekta sa new, vraća se adresa i ona se može pohraniti u varijabli

→ Kada se "object-type" varijabli dodijeli ("assign") nova adresa, vrijednost varijable se mijenja na novu adresu!

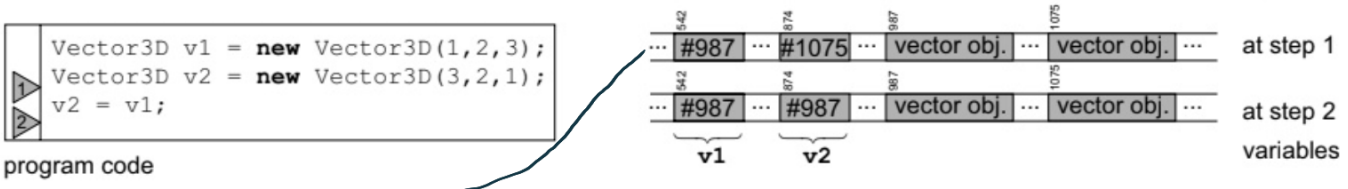


Figure 1.11: Object type variables

→ Kreiran je novi objekat v1 tipa Vector 3D na adresi 542 i V2 na adresi 874

→ Nakon V2=V1 → i V2 pokazuje na isti objekat kao V1

At this point, we introduce another notation for object variables. This new notation makes it easier to visualize the state of an object and connections between objects. As shown in Figure 1.12, an object is visualized as a box where the name of the object's type is underlined. Variables are drawn below the boxes and point to the boxes.

→ novi način označavanja objekta u obliku

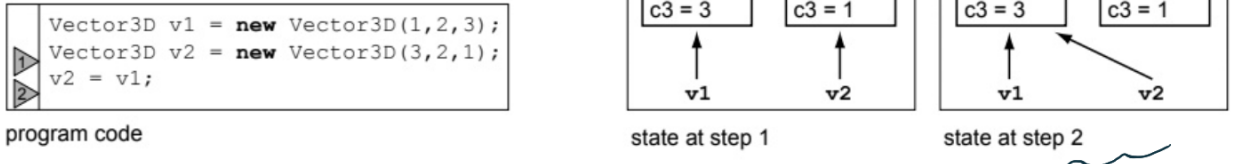


Figure 1.12: New notation for object type variables

objekt variable pocnu pokazivati na isti objekat

2 - Fundamental Programming structures in Java

2.1. Operators

Arithmetic Operators. The arithmetic operators +, −, * and / are used in Java for addition, subtraction, multiplication and division, respectively. The / operator denotes integer division (the decimal part is truncated) if both arguments are integers. Example:

```
double x = 1 / 4;
```

Now x has a value of 0.0 because 1 and 4 both are interpreted as `int`.

If one of the arguments is not an integer, floating-point division is performed. Example:

```
double x = 1.0 / 4;
```

In this case x has a value of 0.25 as expected because 1.0 is interpreted as a `double`. Unintended integer division is a common source of errors.

The integer remainder (the modulus function) is denoted by %. For example;

```
int rem = 15%2;
```

Now `rem` has the value 1, because 15 divided by 2 is 7, remainder 1.

Ostatak ! % → npr. 3 % 2 = 1

13 % 6 = 3

Increment and Decrement Operators. The increment operator ++ adds 1 to a variable; the decrement operator − subtracts 1 from a variable. Example:

```
1 int n = 10;
2 double x = 6.3;
3 n--;
4 x++;
```

Now n has the value 9 and x the value 7.3.

↳ pri svakom prolasku programa prema te ćnje koda, povećaj ili smanji vrijednost za 1

Relational Operators. The relational operators ==, !=, <, <=, > and >= take numerical values and evaluate to a boolean. To test for equality or inequality, use the == or != operator, respectively. For example, the value of

```
4.3 != 5
```

is `true` whereas

```
9%2 == 0
```

evaluates to `false` (this is actually a test whether 9 is an even number). The other relational operators work similarly and their meaning should be self explanatory.

) prođenju itj. poredе dvije numeričke vrijednosti i kao rezultat poređenja dobiju boolean vrijednosti (true - poređenje istinito, false - poređenje nije ispravno)

Boolean Operators. The two boolean operators `&&` (logical *and*) and `||` (logical *or*) take two boolean arguments and evaluate to a boolean. Examples: The value of

```
(5 < 6) && (8%2 == 0)
```

is `true` (both arguments are true) and the value of

```
(7%2 == 0) || (8 <= 8 )
```

is also `true` because one of the arguments is `true`.

) poredе dvije boolean vrijednosti i kao rezultat dobiju boolean evaluaciju poređenja

Table 2.1: Commonly and seldomly used operators in Java

Commonly used operators			
=	simple assignment operator	+	additive operator (also used for String concatenation)
-	subtraction operator	*	multiplication operator
/	division operator	%	remainder operator
++	increment operator	--	decrement operator
==	equal to	!=	not equal to
>	greater than	>=	greater than or equal to
<	less than	<=	less than or equal to
!	logical not	&&	logical and
||	logical or		

ostatak

Seldomly used operators			
~	unary bitwise complement	<<	signed left shift
>>	signed right shift	>>>	unsigned right shift
&	bitwise AND	^	bitwise exclusive OR
|	bitwise inclusive OR	? :	ternary (if-then-else)
instanceof	Compares an object to a specified type		

bitwise