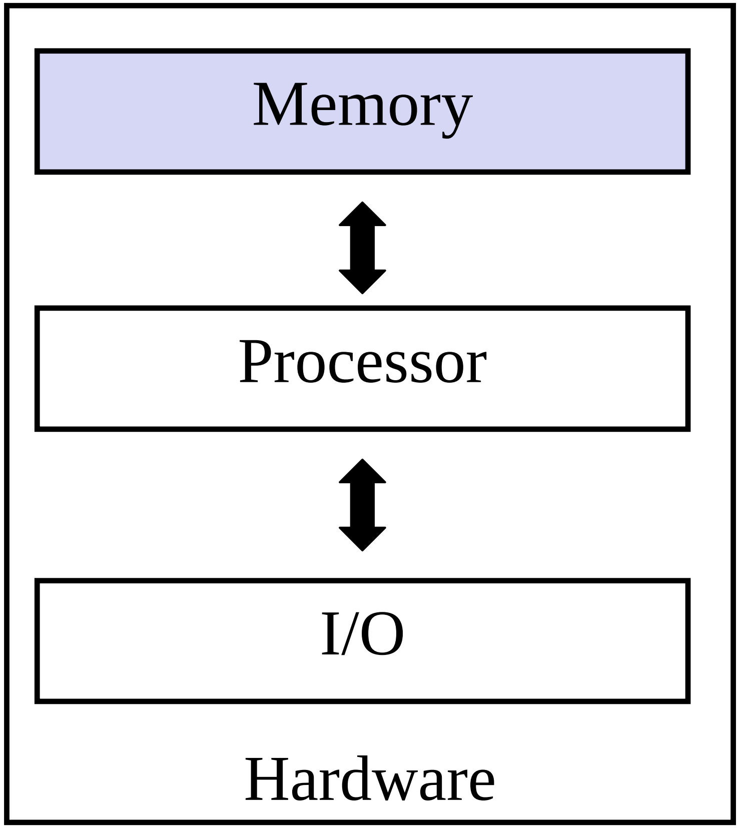
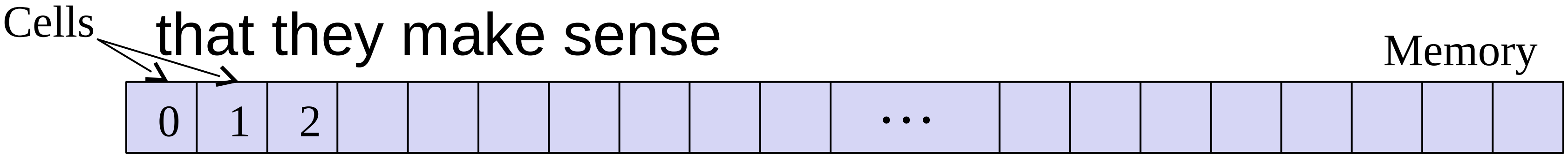
2. Elementary Data Types

Prof. Dr. Charles Wüthrich
B.Sc. Francesco Andreussi
CoGVis/MMC, Faculty of Media
Bauhaus-University Weimar

Memory Organization

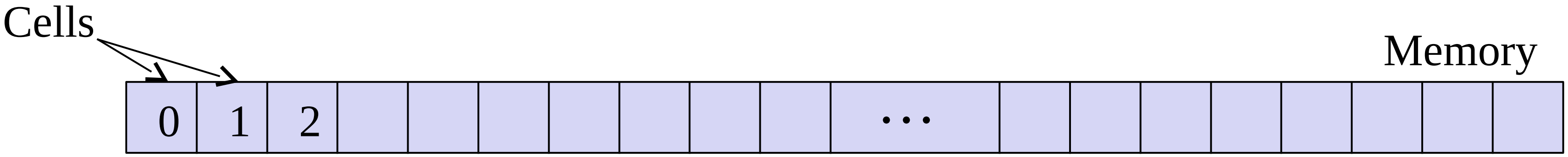
- In the last lesson, we showed a rough scheme of a computer
- Memory is nothing else than a looooooong sequence of "memory cells"
- For the computer, they are unorganized
- It is up to the program-mer to structure it so that they make sense





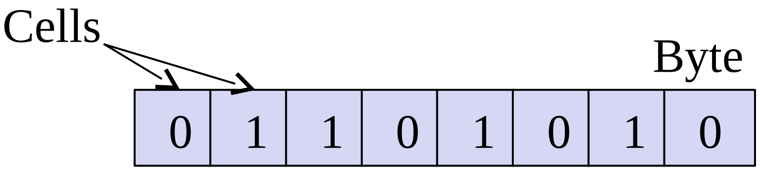
Bits

- Each cell is a ***bit***
  - Smallest unit in a computer
- Computers are basically electrical devices
  - A cell can be either charged (+) or uncharged (-)
  - Or, if you prefer, positively (+) or negatively charged (-)
- Computers know how to understand “charged” and “uncharged” cells
- Ultimately they assign a “number” to the two possible states:
  - 1 for “charged” (alternatively, 1 can be interpreted as TRUE)
  - 0 for “uncharged” (and this as FALSE)



Bytes

- Bits are grouped in ***bytes***
  - Bytes are a group of 8 bits
  - 8 cells
- They can represent 2^8=256 different “things”.
- For example, an integer number between 0 and 255
- The number below is the decimal number 106
- Why?



Bytes

- Bits are grouped in ***bytes***
  - Bytes are a group of 8 bits
  - 8 cells
- They can represent 2^8=256 different “things”.
- For example, an integer number between 0 and 255
- The number below is the decimal number 106
- Why?
  Because $2^6+2^5+2^3+2^1$=64+32+8+2=106
- Bytes are the minimal addressable unit of memory

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	1	1	0	1	0	1	0	Byte

Hexadecimal numbers

- The numbers representable in a byte are often represented not as binary numbers (too long)
- Convenently, as a pair of ***hexadecimal numbers***: as two numbers with a base of
  16 (2^4)=2 groups of 4 bits

Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Byte
0	1	1	0	1	0	1	0	Bin
6				A				Hex

Basic Data Types

- If all the bytes would be binary numbers then only numbers would be usable on a computer
- But you can do more than use numbers on a computer
  - For example, read a Web page, which is not only made by numbers.
  - Or display a picture, that is made of pixels and colours.

- Java allows some basic Data Types off the shelf to introduce variety in what you are able to use and process

Variable declaration

- Most modern programming languages require the programmer to declare a variable and its Data Type
- This to allow the compiler to
  - Reserve the appropriate memory for the variable
  - Know what kind of operations are permitted with this variables
- Notice:
  - Data
  - Operations
- In Object Oriented Programming we call this
  - Data
  - Methods

Boolean

- A Boolean variable can have only two values
  - TRUE
  - FALSE
- It is declared through the following snippet of code:

```
boolean somevariablename;
```

- When the compiler reads this, it
  - Reserves space in memory for a boolean variable
  - Sets itself a reminder that whenever it encounters the word `somevariable` it refers to this particular memory location
  - Remembers that for these memory locations it has to use the operations used for booleans.
- Alternatively, one can assign direct a value when declaring a boolean:

```
boolean somevariablename=FALSE;
```

Boolean

- Which operations are allowed on booleans?

And

AND/∧	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

Or

OR/∨	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

XOR

XOR	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE

Not

VALUE	TRUE	FALSE
NOT/!	FALSE	TRUE

Booleans will be useful later on

Characters

- A *character* is
  - One letter of the alphabet
  - One digit of a number
  - One punctuation sign
  - Any symbol you can type on a keyboard
- It is declared through the following snippet of code:

```
char achar;
```

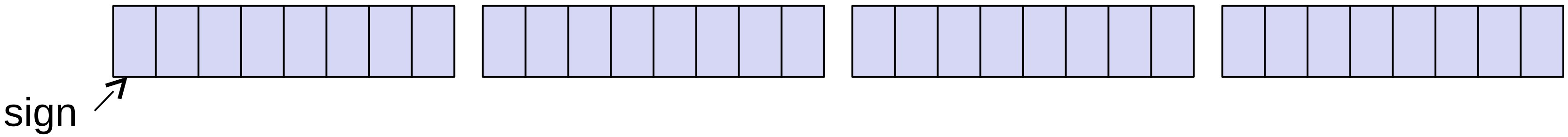
- In this case, the compiler does not assign a value t the reserved cell
- Alternatively, one can assign direct a value when declaring a character:

```
char achar='C';
```

  which assigns the letter C to the character

Integers

- An *integer* in Java is an integer number between -2^{31} and $2^{31}-1$. Its first bit is its sign
- Java reserves 4 bytes for an integer



- It is declared through the following snippet of code:

```
int aninteger;
```

- Allowed operations:

Operation	Symbol
sum	+
difference	-
multiplication	*
division	/
modulo	%

Integers

- Caveat! Division is INTEGER division!

  26/7 = 3!!!

- What on earth is modulo?
  Modulos are rest classes, i.e. the rest of the integer division: 26%7=5
- In math, it a modulo b is indicated as a `mod` b

- One can also have unsigned integers, which are integers between 0 and 2^{32}-1:

```
unsigned int anunsigned;
```

Long Integers

- Sometimes one needs bigger numbers:
  long integers reserve 8 instead of four bytes
- They can therefore store integer numbers between -2^{63} and $2^{63}-1$
- They are declared as follows:

```
long along;
```

- All operations allowed in integers are allowed also on longs
- Why not use directly long integers? They take up double space
- Also here, you can declare `unsigned long along;`

Floating point numbers

- *Floating point* numbers represent your numbers as
  - The sign
  - A binary exponent between -127 and 126
  - The mantissa, which is between 0 and 1
  - The bitwise representation looks as follows:

SEEEEEEEE	EMMMMMMMM	MMMMMMMMM	MMMMMMMMM

    where
    - S sign
    - E exponent
    - M mantissa, or fractional part normalized between 0.5 and 1
  - Resulting number : $(-1)^s * m * 2^{e-127}$

- Declaration is done through the following line:

```
float afloat;
```

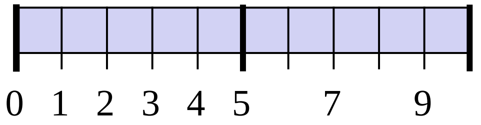
Floating point numbers

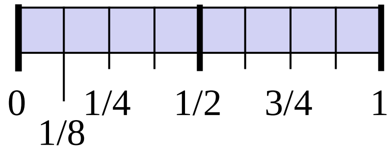
- Assigning numbers to floating points

```
float afloat=2.57f;
```

- So the compiler understands that the number is a float

- HUGE CAVEAT! Intervals are different from decimal subdivisions!!!





Double precision numbers

- *Double precision* numbers are similar, but use 8 bytes for your numbers
  - One bit for the sign
  - A binary exponent with 11 bits
  - The mantissa, which is between 0 and 1 and has 52 bits

SEEEEEEE EEEEMMMM MMMMMMM MMMMMMMM MMMMMMMM MMMMMMMM MMMMMMMM MMMMMMMM

- Declaration is done through the following line:

```
double adouble;
```

Casting Data Types

- A compiler will NOT mix datatypes
  - Adding an integer with a floating point will give you an error!
- Only way to do it:
  - Casting: explicitly declare the type of your variable to force its conversion

  - If we obtain for a student a grade of 87.6 and the final grade can only be an integer then we can write

```
float calculatedMark = 87.6f;
int finalGrade = (int)calculatedMark;
```

  - Which rounds the final grade to an integer.

Printing Data Types: + form

```
public class PrintVariables {
  public static void main(String[] args) {

    int anint;
    float afloat;
    char achar;
    anint = 10;
    afloat = 32.23f;
    achar = 'C';
    System.out.println("The integer was "+anint);
    System.out.println("The floating point was "+afloat);
    System.out.println("The character was a "+achar);
  } // main
} // PrintVariables
```

Printing Data Types: % form

```
public class PrintVariables {
    public static void main(String[] args) {
        int anint;
        float afloat;
        char achar;
        anint = 10;
        afloat = 32.23f;
        achar = 'C';
        System.out.println("The integer was "+anint);
        System.out.printf("The integer was %d, the float %f, the character %s %n",anint,afloat,achar);
        System.out.println("The floating point was "+afloat);
        System.out.println("The character was a "+achar);
    } // main
} // PrintVariables
```

Strings

- A *String* is a concatenation of characters:

  ```
  This is a string
  ```

- Strings are declared through the following declaration:

  ```
  String astring = “This is a string”
  ```

- The only basic operation on strings is their concatenation “+”

  ```
  String astring1, astring2, astring3;
  astring1 = "Tomorrow";
  astring2 = " is Tuesday.";
  astring3 = astring + astring2;
  ```

  – Which results in

  ```
  Tomorrow is Tuesday.
  ```

Converting Strings

- A *String* can be converted to another Type through the Java methods:

```
int Integer.ParseInt(String astring)
long Long.ParseLong(String astring)
float Float.ParseFloat(String astring)
double Double.ParseDouble(String astring)
```

- “Parse” is the Computer Science word for “Read and understand”

Converting Strings

- Let us take a look at the code:

```
// Takes your name, your integer age and your height from the
// command line and prints them
public class ReadNameAgeHeight {
   public static void main(String[] args) {
      String name;
      int age;
      float height;

      //1st command line paramenter
      name = args[0];

      //2nd command line paramenter
      age = Integer.parseInt(args[1]);

      //3rd command line parameter
      height = Float.parseFloat(args[2]);

      System.out.printf(
        "%s, your age is %d and you are %fm tall%n",name,age,height
        );
   } // main
} // ReadNameAgeHeight
```

Converting Strings

- Let us take a look at the code:

```
public class ReadNameAgeHeight {
   public static void main(String[] args) {
      String name;
      int age;
      float height;

      //1st command line paramenter
      name = args[0];

      //2nd command line paramenter
      age = Integer.parseInt(args[1]);

      //3rd command line parameter
      height = Float.parseFloat(args[2]);

      System.out.printf(
        "%s, your age is %d and you are %fcm
            tall%n",name,age,height
        );
   } // main
} // ReadNameAgeHeight
```

- The output of typing in the shell:
  `> java ReadnameHeight Frodo 135 0,57`

- Will result in the output:

```
Frodo, your age is 135 and
you are 0.570000m tall
```

- Note: the first argument is a string, so it needs no conversion!

End

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++