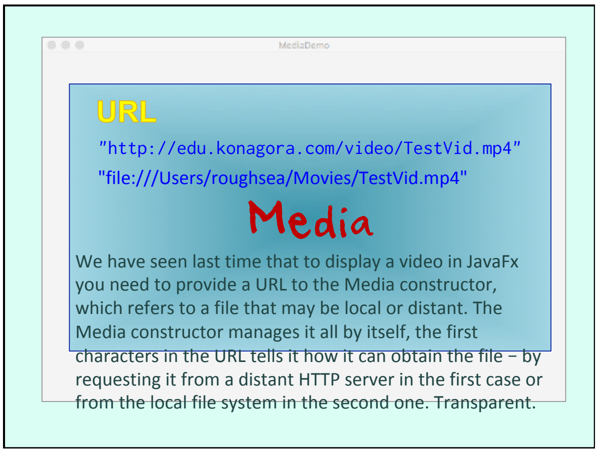
CS209

Computer system design and application

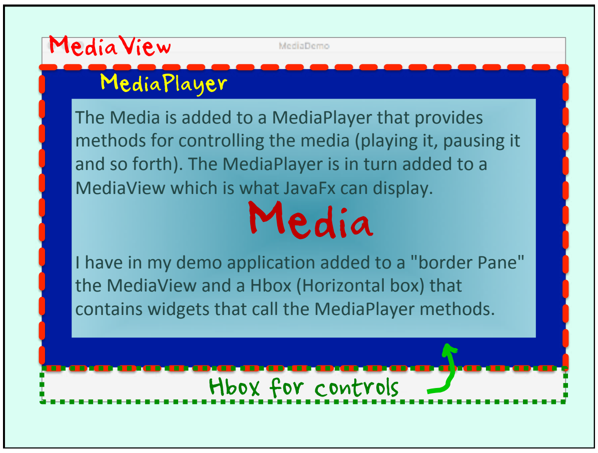
Stéphane Faroult
faroult@sustc.edu.cn

Zhao Yao zhaoy6@sustc.edu.cn
Liu Zijian liuzijian47@163.com
Li Guansong intofor@163.com



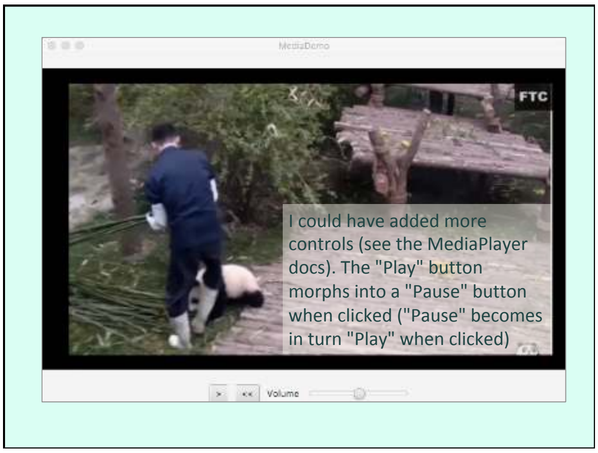
"http://edu.konagora.com/video/TestVid.mp4"
"file:///Users/roughsea/Movies/TestVid.mp4"

We have seen last time that to display a video in JavaFx you need to provide a URL to the Media constructor, which refers to a file that may be local or distant. The Media constructor manages it all by itself, the first characters in the URL tells it how it can obtain the file – by requesting it from a distant HTTP server in the first case or from the local file system in the second one. Transparent.



The Media is added to a MediaPlayer that provides methods for controlling the media (playing it, pausing it and so forth). The MediaPlayer is in turn added to a MediaView which is what JavaFx can display.

I have in my demo application added to a "border Pane" the MediaView and a Hbox (Horizontal box) that contains widgets that call the MediaPlayer methods.



I could have added more controls (see the MediaPlayer docs). The "Play" button morphs into a "Pause" button when clicked ("Pause" becomes in turn "Play" when clicked)

Skinning a GUI using CSS

Last JavaFx subject, superficial in all meanings of the word but important (looks sell): how to change the appearance of a JavaFx application. I have already briefly talked about it, the best way is to do it through an external style sheet (.css file). People will be able to change, often in a very impressive way, the looks of your application by changing this file and without any need to access the code (in fact, they just need the .css and the .class to run the "modified" application).
If you want to see how far you go with "styling", you can visit http://csszengarden.com and click on designs on the right hand-side. The same page will look completely different.

I have already said it, CSS stands for Cascading Style Sheet and Cascade is French for waterfall.
The name emphasizes that styles "drip down", and are inherited from parent structures.

Cascading
Style
Sheet

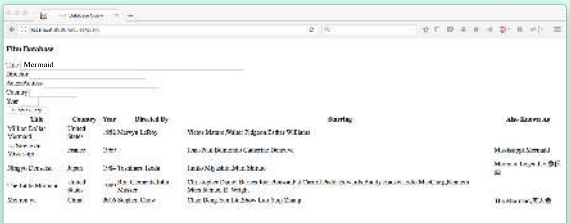
It all starts with HTML

HyperTextMarkupLanguage

The idea is stolen from web applications. Web applications just send HTML pages to browsers that decode and display these pages. An HTML page is organized by sections between pairs of tags (<tag> at the beginning, </tag> at the end) that structure the document and can contain in turn other pairs of tags, thus defining a kind of hierarchical structure (note that some tags, such as those for images, act both as opening and closing tags). Tags pretty often also contain attributes (such as the image file name for an image tag).

<tags> ← Associate formatting with each tag in a "stylesheet"

In the very early days of the web, people were using tags to format their pages, for instance what they wanted in bold was between **<bold>** and **</bold>** (inspired by previous document generation systems that were sending special signals to printers), and you could change the fonts with **<font** *attributes specifying the font, size and everything***>** ... **</font>**. As websites were growing in size and number of pages, and as increasingly pages were generated by programs instead of being created by hand, it became unmanageable, especially when the marketing department was deciding on new corporate colors. So the idea was to associate formatting to tags in one or several separate text files.

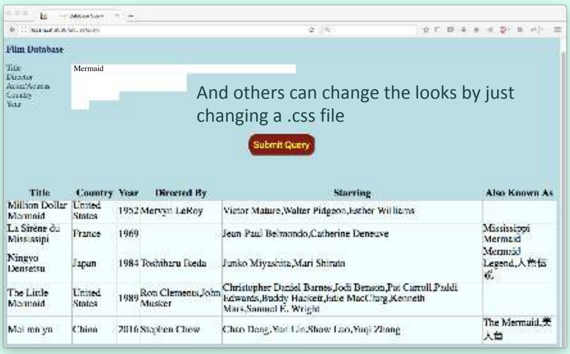


Film Database

Title	Mermaid
Director	
Actor/Actress	
Country	
Year	

Title	Country	Year	Directed By	Starring	Also Known As
Million Dollar Mermaid	United States	1952	Mervyn LeRoy	Victor Mature,Walter Pidgeon,Esther Williams	
La Sirène du Mississipi	France	1969		Jean Paul Belmondo,Catherine Deneuve	Mississippi Mermaid
Ningyo Densetsu	Japan	1984	Toshiharu Ikeda	Junko Miyashita,Mari Shirato	Mermaid Legend,人魚伝説
The Little Mermaid	United States	1989	Ron Clements,John Musker	Christopher Daniel Barnes,Jodi Benson,Pat Carroll,Paddi Edwards,Buddy Hackett,Jason Marin,Kenneth Mars,Samuel E. Wright	
Mei ren yu	China	2016	Stephen Chow	Chao Deng,Yun Lin,Show Luo,Yuqi Zhang	The Mermaid,美人鱼

Some people could just focus on the "engine", such as here an application allowing to retrieve film information from a database ...



Film Database

Title	Mermaid
Director	
Actor/Actress	
Country	
Year	

And others can change the looks by just changing a .css file

Title	Country	Year	Directed By	Starring	Also Known As
Million Dollar Mermaid	United States	1952	Mervyn LeRoy	Victor Mature,Walter Pidgeon,Esther Williams	
La Sirène du Mississipi	France	1969		Jean Paul Belmondo,Catherine Deneuve	Mississippi Mermaid
Ningyo Densetsu	Japan	1984	Toshiharu Ikeda	Junko Miyashita,Mari Shirato	Mermaid Legend,人魚伝説
The Little Mermaid	United States	1989	Ron Clements,John Musker	Christopher Daniel Barnes,Jodi Benson,Pat Carroll,Paddi Edwards,Buddy Hackett,Jodie MacClurg,Kenneth Mars,Samuel E. Wright	
Mei ren yu	China	2016	Stephen Chow	Chao Deng,Yun Lin,Show Luo,Yuqi Zhang	The Mermaid,美人鱼

(I know it's ugly but I didn't work very hard on it)

The way it works in web pages

Before I discuss about CSS in JavaFx, I'm going to talk about CSS with HTML, because there is far more CSS written for HTML than for JavaFX.

There are three main ways to specify how to display what is between tags.
 1. You can specify for a given tag, associating a tagname with visual characteristics
2. I have mention that tags can have attributes, one is "class" (unrelated to object-oriented programming) listing one or several categories. This allows to create a subcategory, or to give some common visual characteristics across different tags that have the same class (for instance "inactive")
3. You can give another attribute "id" to a tag, an this allows you to make one particular tag look really special.

```
<tag>            tag {
                    attribute: value;
                       ...
                 }
```

I focus on simple tags in the next example.

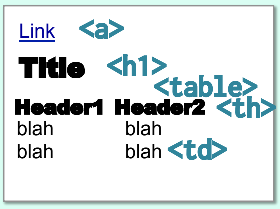
```
<tag class="category">
                 .category {
                    attribute: value;
                       ...
                 }
```

You have here how the tag looks in HTML and how styling looks in CSS.

```
<tag id="name">
                 #name {
                    attribute: value;
                       ...
                 }
```

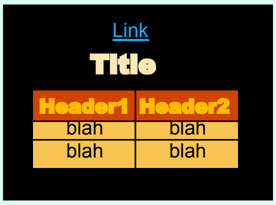
<body>

In a HTML page, everything that is displayed is between <body> and </body> tags, so <body> really defines the global looks of the page.



In my example, I'll use tags <a> associated with a link, <h1> with a (first level) header, <table>, <th> (table header) and <td> (table data). I have omitted <tr> (table row) which usually surrounds the columns of a same row. The tag will be there on the page, but I'm not associating any special style with it in my example.

```
body {text-align: center;
      background-color: black;}
a {color: lightskyblue;}
a:visited {color: lightseagreen;}
a:hover {color: salmon;}
h1 {color: cornsilk;}
th {background-color: sienna;
    color: orange;}
td {background-color: burlywood;}
table {width: 80%;
       margin-left: auto;
       margin-right: auto;
       text-align: center;}
form {width: 50%;
      margin-left: auto;
      margin-right: auto;
      padding: 20px;
      background-color: silver;}
```



css color names

This will give you the names of colors understood by browsers.

Notice the special a:visited (link you have already clicked on) and a:hover (when the cursor moves over the link)

```
body {text-align: center;
      background-color: black;}
a {color: lightskyblue;}
a:visited {color: lightseagreen;}
a:hover {color: salmon;}
h1 {color: cornsilk;}
th {background-color: sienna;
    color: orange;}
td {background-color: burlywood;}
table {width: 80%;
       margin-left: auto;
       margin-right: auto;
       text-align: center;}
form {width: 50%;
      margin-left: auto;
      margin-right: auto;
      padding: 20px;
      background-color: silver;}
```

styles.css

Styling is written to a file that is included in the HTML page by a special tag in the <header>...</header> section that precedes the "body". Then you can change it when needed.

```
<link rel="stylesheet" href="styles.css"/>
```

The way it works in javaFX

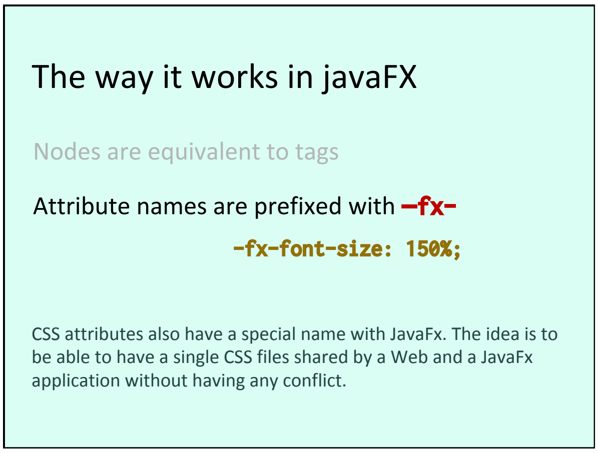
Nodes are almost equivalent to tags

.root plays the same role as **body**

otherwise use class names prefixed with a dot

.button

You have of course no tags in a JavaFx application, but you have the same kind of hierarchy through nodes, containers and widgets. The JavaFx class names are used with the same syntax as the HTML classes in CSS, prefixed by a dot.



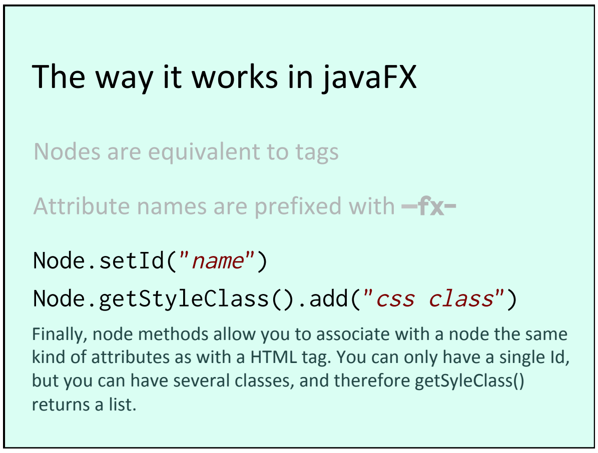
The way it works in javaFX

Nodes are equivalent to tags

Attribute names are prefixed with **-fx-**

```
-fx-font-size: 150%;
```

CSS attributes also have a special name with JavaFx. The idea is to be able to have a single CSS files shared by a Web and a JavaFx application without having any conflict.



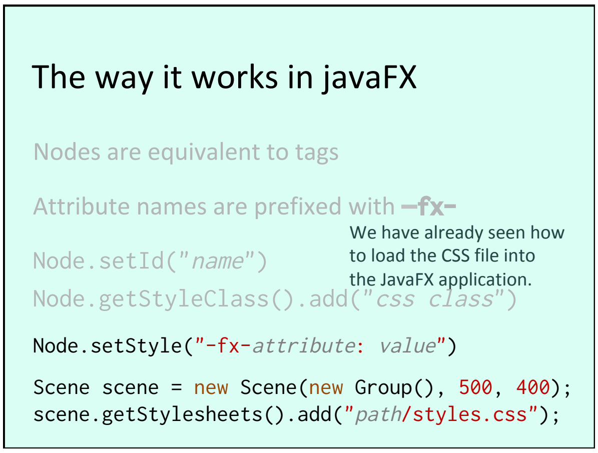
The way it works in javaFX

Nodes are equivalent to tags

Attribute names are prefixed with **-fx-**

```
Node.setId("name")
Node.getStyleClass().add("css class")
```

Finally, node methods allow you to associate with a node the same kind of attributes as with a HTML tag. You can only have a single Id, but you can have several classes, and therefore getSyleClass() returns a list.



The way it works in javaFX

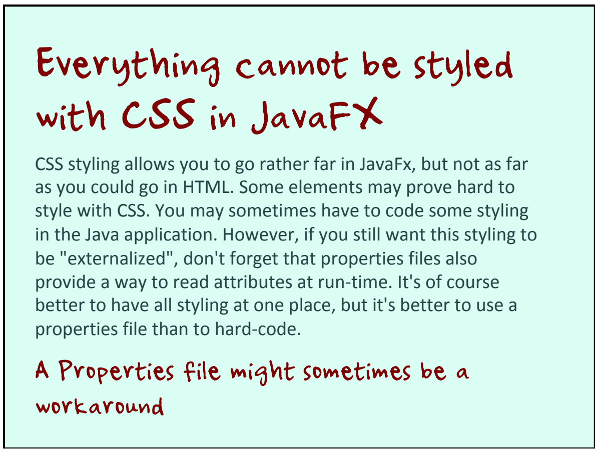
Nodes are equivalent to tags

Attribute names are prefixed with **-fx-**

```
Node.setId("name")
Node.getStyleClass().add("css class")
```

We have already seen how to load the CSS file into the JavaFX application.

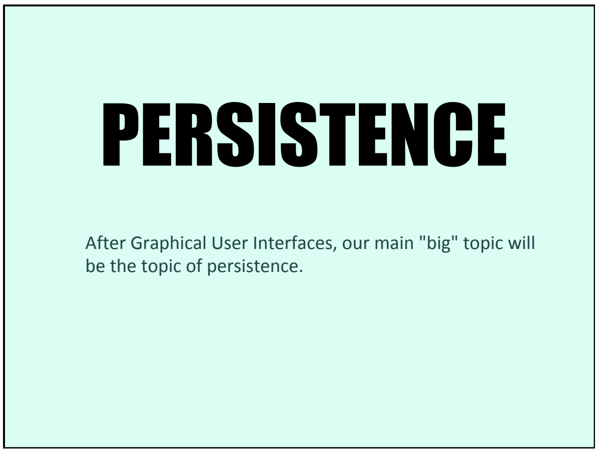
```
Node.setStyle("-fx-attribute: value")

Scene scene = new Scene(new Group(), 500, 400);
scene.getStylesheets().add("path/styles.css");
```



Everything cannot be styled with CSS in JavaFX

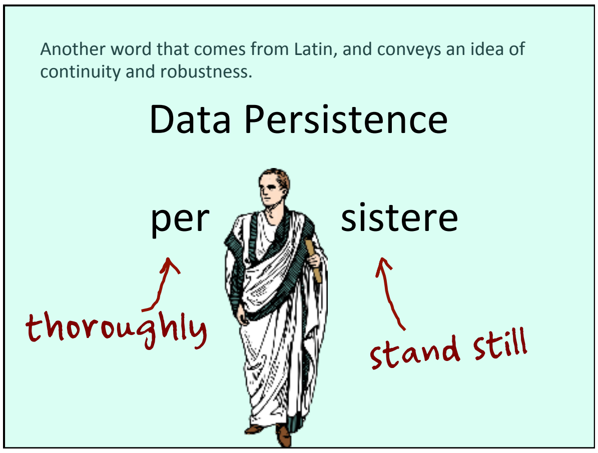
CSS styling allows you to go rather far in JavaFx, but not as far as you could go in HTML. Some elements may prove hard to style with CSS. You may sometimes have to code some styling in the Java application. However, if you still want this styling to be "externalized", don't forget that properties files also provide a way to read attributes at run-time. It's of course better to have all styling at one place, but it's better to use a properties file than to hard-code.

A Properties file might sometimes be a workaround



PERSISTENCE

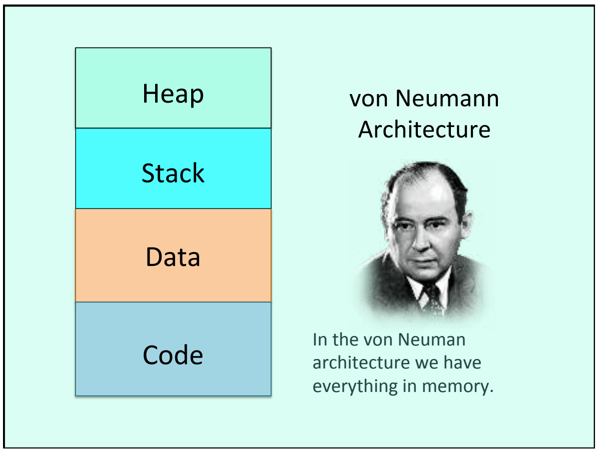
After Graphical User Interfaces, our main "big" topic will be the topic of persistence.



Another word that comes from Latin, and conveys an idea of continuity and robustness.

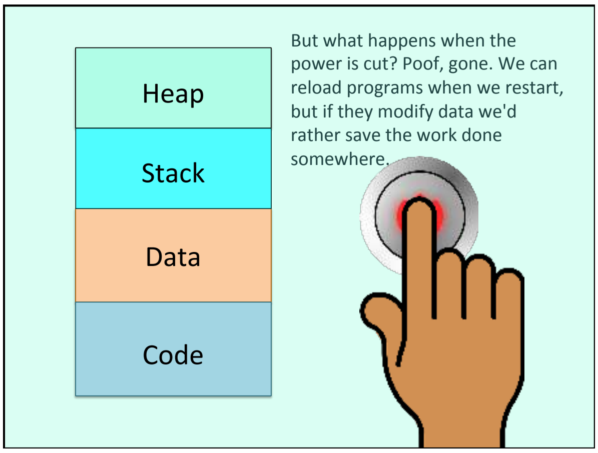
Data Persistence

per
sistere
thoroughly
stand still

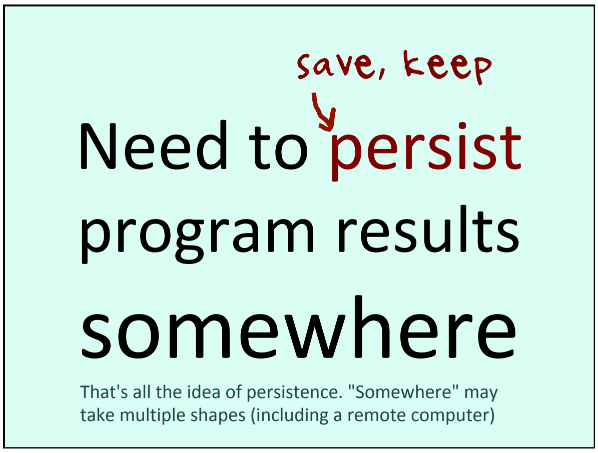


von Neumann Architecture

In the von Neuman architecture we have everything in memory.



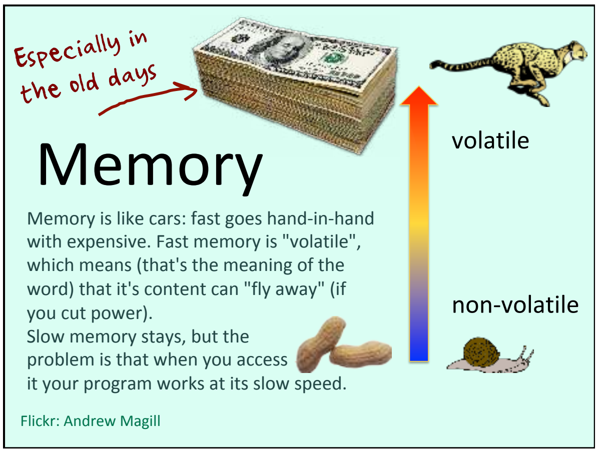
But what happens when the power is cut? Poof, gone. We can reload programs when we restart, but if they modify data we'd rather save the work done somewhere.



save, keep

Need to persist program results somewhere

That's all the idea of persistence. "Somewhere" may take multiple shapes (including a remote computer)



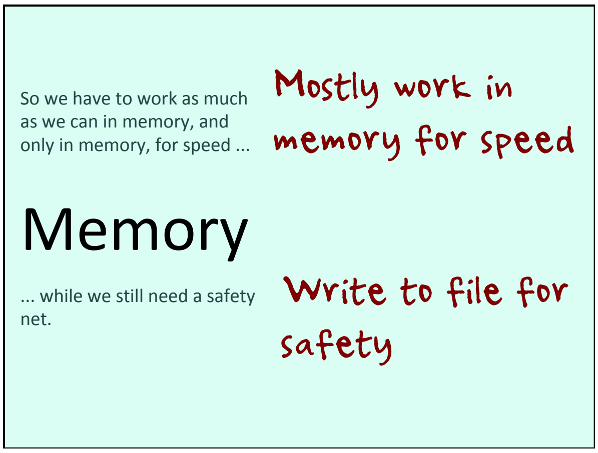
Especially in the old days

Memory

Memory is like cars: fast goes hand-in-hand with expensive. Fast memory is "volatile", which means (that's the meaning of the word) that it's content can "fly away" (if you cut power).
Slow memory stays, but the problem is that when you access it your program works at its slow speed.

volatile
non-volatile

Flickr: Andrew Magill



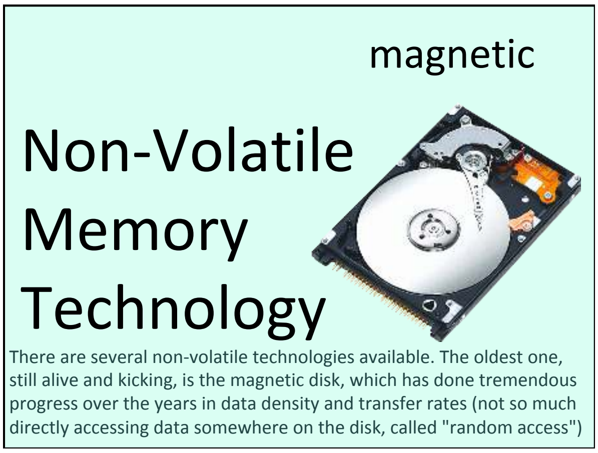
So we have to work as much as we can in memory, and only in memory, for speed ...

Mostly work in memory for speed

Memory

... while we still need a safety net.

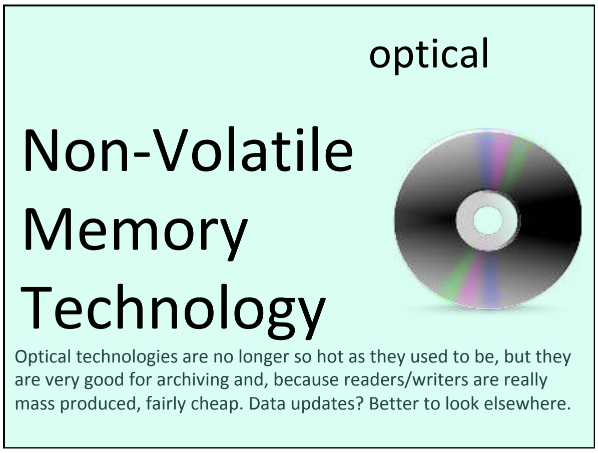
Write to file for safety



magnetic

Non-Volatile Memory Technology

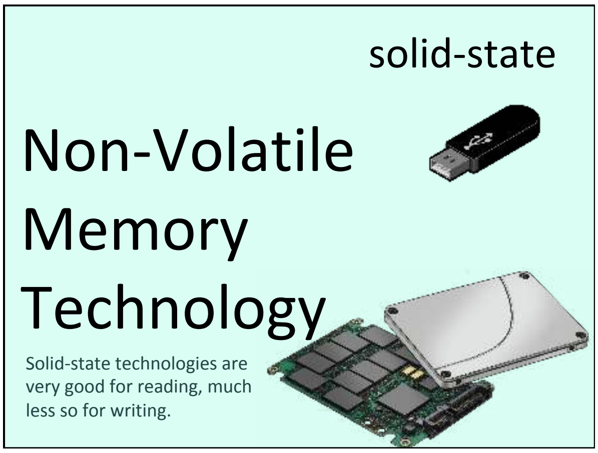
There are several non-volatile technologies available. The oldest one, still alive and kicking, is the magnetic disk, which has done tremendous progress over the years in data density and transfer rates (not so much directly accessing data somewhere on the disk, called "random access")



optical

Non-Volatile Memory Technology

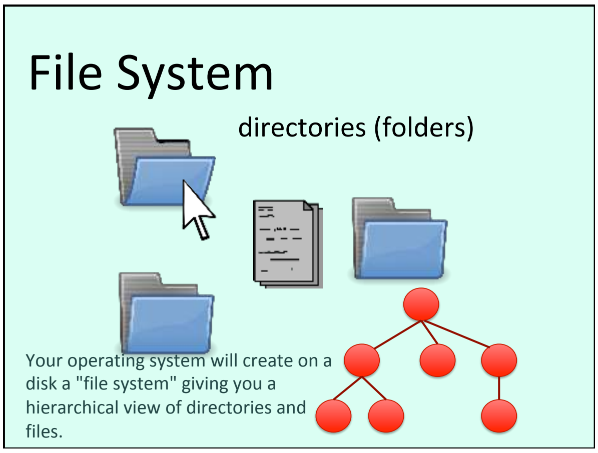
Optical technologies are no longer so hot as they used to be, but they are very good for archiving and, because readers/writers are really mass produced, fairly cheap. Data updates? Better to look elsewhere.



solid-state

Non-Volatile Memory Technology

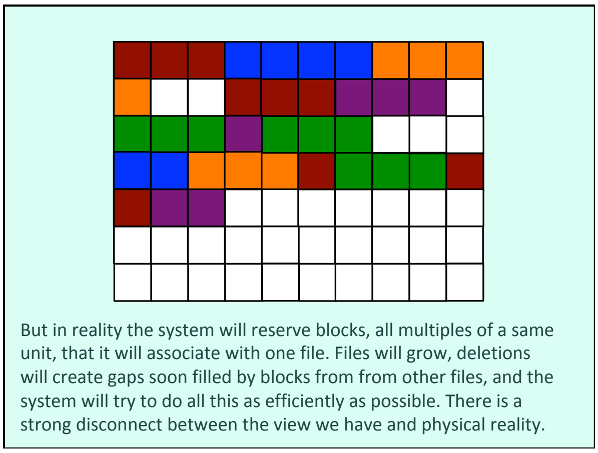
Solid-state technologies are very good for reading, much less so for writing.



File System

directories (folders)

Your operating system will create on a disk a "file system" giving you a hierarchical view of directories and files.



🟥	🟥	🟥	🟦	🟦	🟦	🟦	🟧	🟧	🟧
🟧	⬜	⬜	🟥	🟥	🟥	🟪	🟪	🟪	⬜
🟩	🟩	🟩	🟪	🟩	🟩	🟩	⬜	⬜	⬜
🟦	🟦	🟧	🟧	🟧	🟥	🟩	🟩	🟩	🟥
🟥	🟪	🟪	⬜	⬜	⬜	⬜	⬜	⬜	⬜
⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜
⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜	⬜

But in reality the system will reserve blocks, all multiples of a same unit, that it will associate with one file. Files will grow, deletions will create gaps soon filled by blocks from from other files, and the system will try to do all this as efficiently as possible. There is a strong disconnect between the view we have and physical reality.

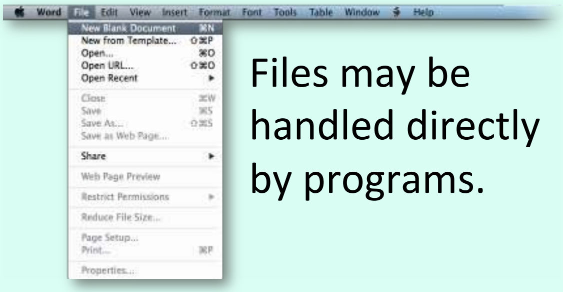
Stream redirection

Instead of keyboard

```
$ java MyProgram < input_file

$ java MyProgram > output_file
```

Instead of screen

One very easy way to save data and to restore it is, instead of doing it in Java, to let the system do it through what is known as "Stream redirection". Your program may read from what it thinks is the keyboard when in reality input comes from a file, and what is written to the screen can also be redirected to a file.



Files may be handled directly by programs.

Many program, though, will directly handle file either when you specifically request it, or in your back (for instance "autosave" every ten minutes).

Two traditional types of files:

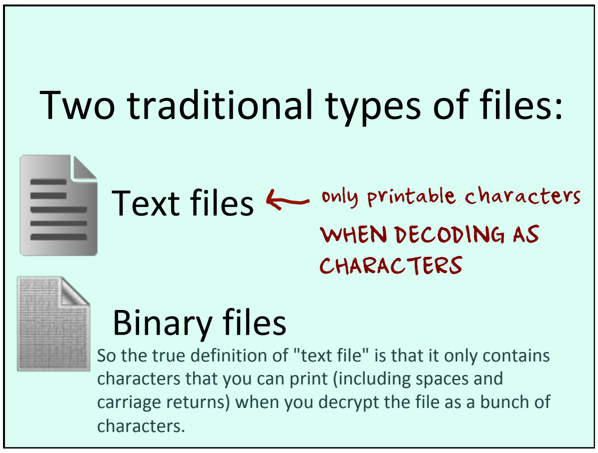
Text files ← only printable characters

Binary files

If there are tons of file extensions and many types of files, they all fall in one of two categories: text or binary. The only problem is, whatever they are, they are all made of 0s and 1s. So really it's all a matter of interpretation.

Interpretation is the big, hard question. You cannot guess the meaning of 1s and 0s just by looking at them. You must have an idea already. And even with text, there are many different ways to encode one single character (and don't believe that the problem doesn't exist even with basic Latin letters – there is another encoding system than ASCII called EBCDIC and the bits meaning 'a' in ASCII mean '/' in EBCDIC). If you haven't the key allowing you to decrypt the bits, you are lost.

HOW to understand the 0's and 1's?

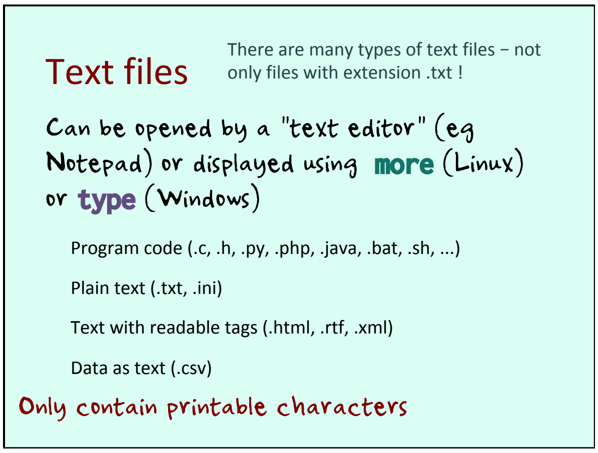


Two traditional types of files:

Text files ← only printable characters WHEN DECODING AS CHARACTERS

Binary files

So the true definition of "text file" is that it only contains characters that you can print (including spaces and carriage returns) when you decrypt the file as a bunch of characters.



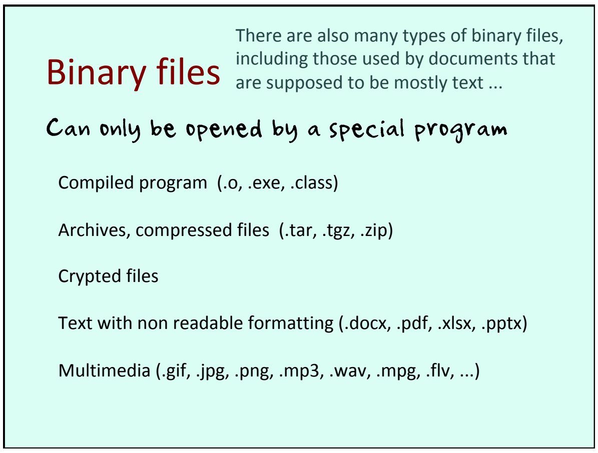
Text files

There are many types of text files – not only files with extension .txt !

Can be opened by a "text editor" (eg Notepad) or displayed using **more** (Linux) or **type** (Windows)

- Program code (.c, .h, .py, .php, .java, .bat, .sh, ...)
- Plain text (.txt, .ini)
- Text with readable tags (.html, .rtf, .xml)
- Data as text (.csv)

Only contain printable characters

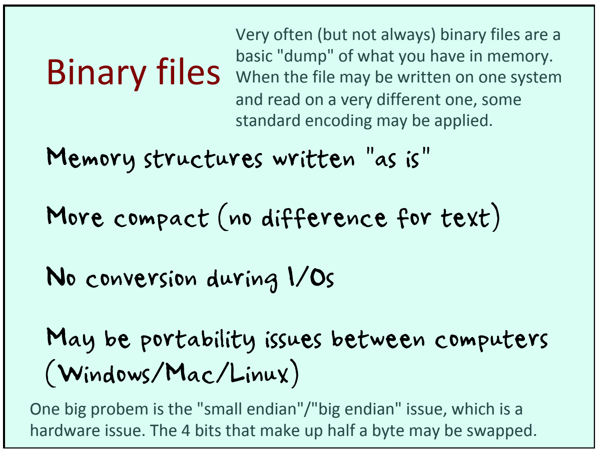


Binary files

There are also many types of binary files, including those used by documents that are supposed to be mostly text ...

Can only be opened by a special program

- Compiled program (.o, .exe, .class)
- Archives, compressed files (.tar, .tgz, .zip)
- Crypted files
- Text with non readable formatting (.docx, .pdf, .xlsx, .pptx)
- Multimedia (.gif, .jpg, .png, .mp3, .wav, .mpg, .flv, ...)



Binary files

Very often (but not always) binary files are a basic "dump" of what you have in memory. When the file may be written on one system and read on a very different one, some standard encoding may be applied.

Memory structures written "as is"

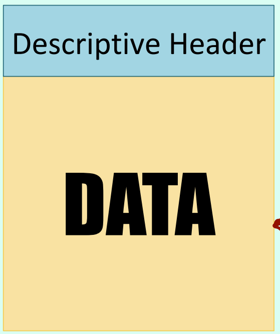
More compact (no difference for text)

No conversion during I/Os

May be portability issues between computers (Windows/Mac/Linux)

One big probem is the "small endian"/"big endian" issue, which is a hardware issue. The 4 bits that make up half a byte may be swapped.

Binary files



Bunch of attributes

Most often a binary file contains a header part that describes the structure, followed by data proper.

Bunch of 0s and 1s

Stream redirection

Text files (most often)

```
$ java MyProgram < input_file

$ java MyProgram > output_file
```

Although nothing forbids writing to/reading from binary files, stream redirection is mostly used with text files.

What is important to understand is that whether you are calling println with an integer, whether you are explicitly calling toString() or not, a conversion occurs.

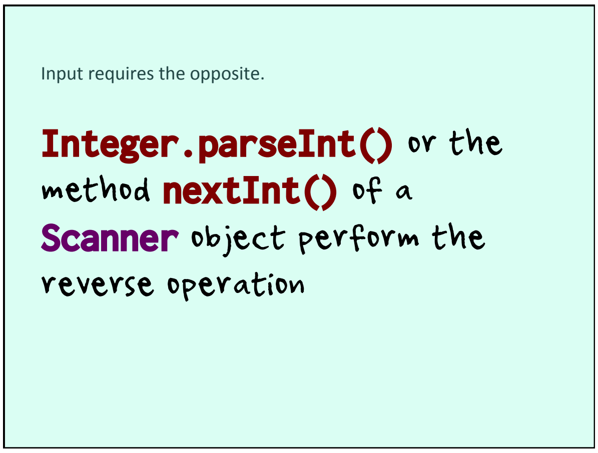
digits (characters)

```
Integer.toString(int_val);
```

Binary internal computer representation

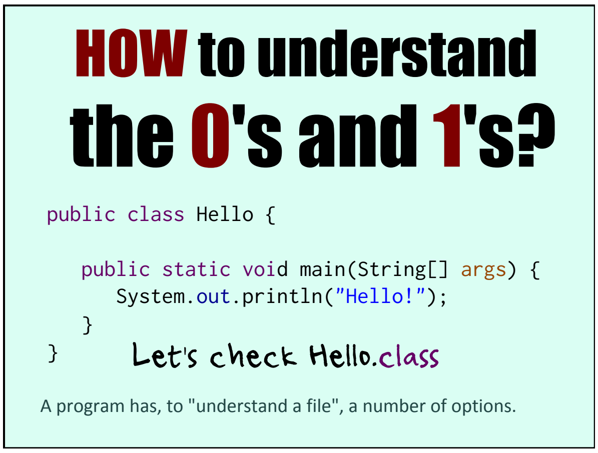
The number has to be turned into a string of digits for output.

```
if number is negative
   display '-'
loop on decreasing powers of 10
   get the result r of the integer division of the number by the power of 10
   if we have already displayed a non zero digit
      display the digit corresponding to the code of '0' plus r
   else
      if the digit is not zero
         record that we have found a non zero digit
         display the digit corresponding to the code of '0' plus r
   decrease the number by r times the power of 10 processed
end of loop
```



Input requires the opposite.

Integer.parseInt() or the method **nextInt()** of a **Scanner** object perform the reverse operation

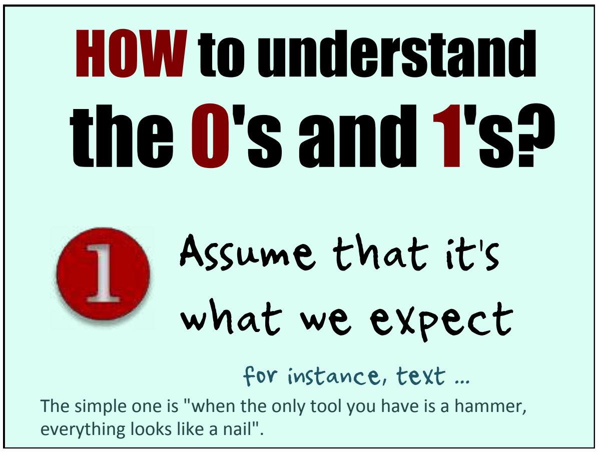


HOW to understand the 0's and 1's?

```
public class Hello {

   public static void main(String[] args) {
      System.out.println("Hello!");
   }
}
```

Let's check Hello.class

A program has, to "understand a file", a number of options.

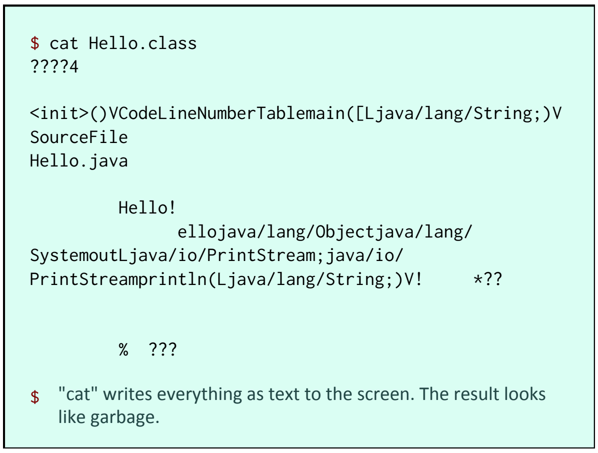


HOW to understand the 0's and 1's?

1 Assume that it's what we expect

for instance, text ...

The simple one is "when the only tool you have is a hammer, everything looks like a nail".



```
$ cat Hello.class
????4

<init>()VCodeLineNumberTablemain([Ljava/lang/String;)V
SourceFile
Hello.java

         Hello!
               ellojava/lang/Objectjava/lang/
SystemoutLjava/io/PrintStream;java/io/
PrintStreamprintln(Ljava/lang/String;)V!     *??


         %  ???

$
```

"cat" writes everything as text to the screen. The result looks like garbage.

HOW to understand the 0's and 1's?

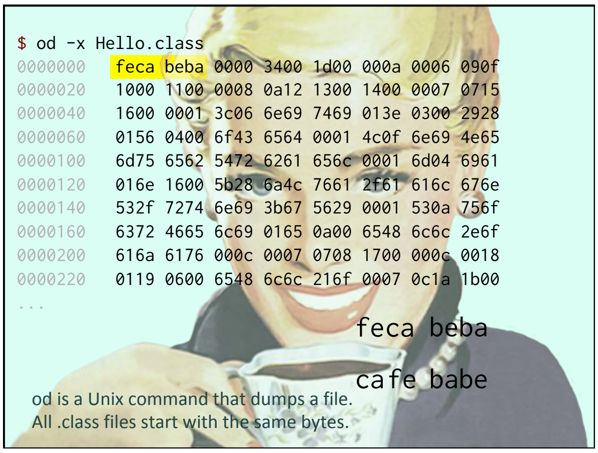
2 Assume that the extension is correct

I renamed Hello.class to Hello.c and tried to compile it. I got 105 warnings and 11 errors but the compiler tried.

HOW to understand the 0's and 1's?

3 Check the file header for a "magic number"

Binary files usually contain a "signature" in their header, a small number of bytes that are very specific to one type of files. You don't need to trust the extension.



```
$ od -x Hello.class
0000000   feca beba 0000 3400 1d00 000a 0006 090f
0000020   1000 1100 0008 0a12 1300 1400 0007 0715
0000040   1600 0001 3c06 6e69 7469 013e 0300 2928
0000060   0156 0400 6f43 6564 0001 4c0f 6e69 4e65
0000100   6d75 6562 5472 6261 656c 0001 6d04 6961
0000120   016e 1600 5b28 6a4c 7661 2f61 616c 676e
0000140   532f 7274 6e69 3b67 5629 0001 530a 756f
0000160   6372 4665 6c69 0165 0a00 6548 6c6c 2e6f
0000200   616a 6176 000c 0007 0708 1700 000c 0018
0000220   0119 0600 6548 6c6c 216f 0007 0c1a 1b00
...
```

feca beba

cafe babe

od is a Unix command that dumps a file.
All .class files start with the same bytes.

Most file-related classes are in the java.io package (there is also a java.nio). Two types of streams, Character or Byte, which can all be buffered or unbuffered.

Handling files in Java

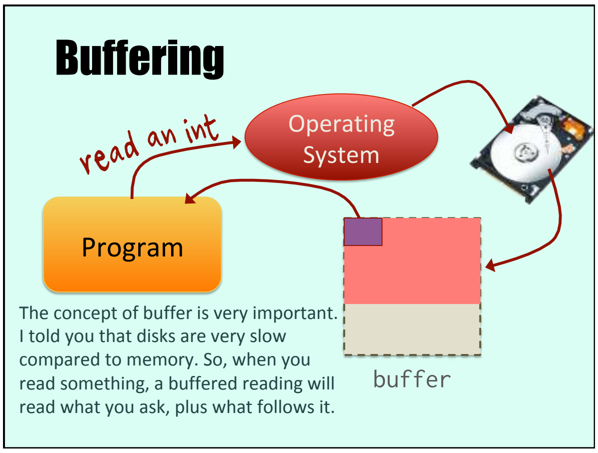
Key concept: Stream

Character streams

Buffered or unbuffered

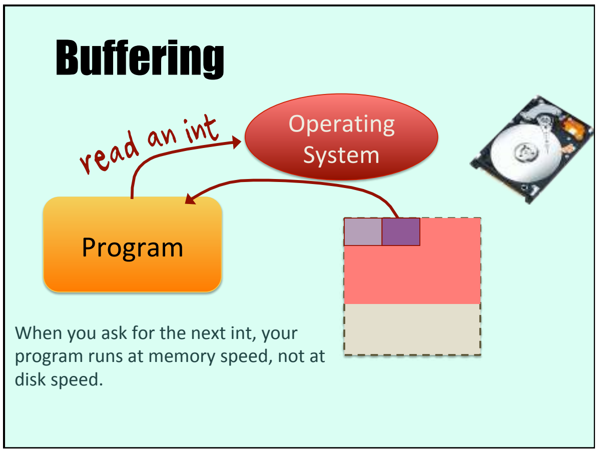
Byte streams

```
import java.io.*
```



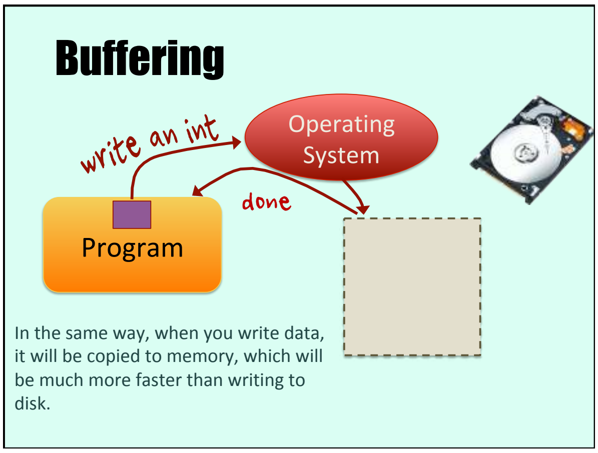
Buffering

The concept of buffer is very important. I told you that disks are very slow compared to memory. So, when you read something, a buffered reading will read what you ask, plus what follows it.



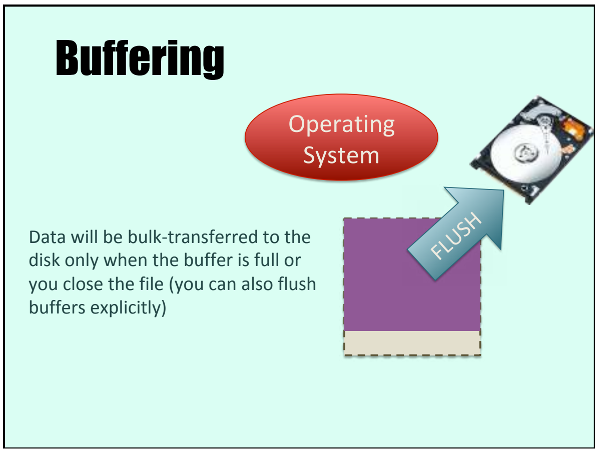
Buffering

When you ask for the next int, your program runs at memory speed, not at disk speed.



Buffering

In the same way, when you write data, it will be copied to memory, which will be much more faster than writing to disk.



Buffering

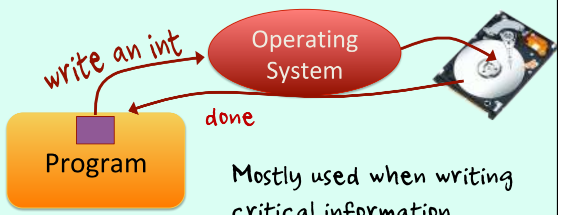
Data will be bulk-transferred to the disk only when the buffer is full or you close the file (you can also flush buffers explicitly)

Buffering

What happens if the system restarts after a crash?

The answer of course is that what was in buffers is lost. Not a problem when reading, big problem when writing. It's not always easy to know what to replay.

No buffering



Mostly used when writing critical information

Logs

Messaging

If you freak about crashes, you should use slow unbuffered operations.

Performance - copying a 11M CSV file character by character

Test on my Mac (internal SSD)

Unbuffered	about 34.5s
Buffered	about 1s

Test on my Mac (External USB HD)

Unbuffered	about 36s
Buffered	about 1.4s

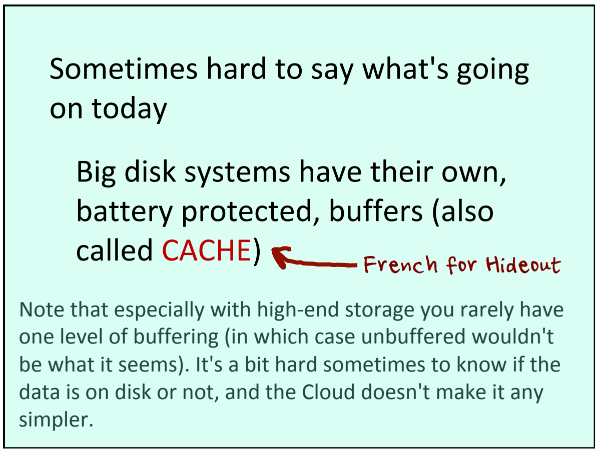
How often does your computer crash? Would it be a complete disaster to run the program again after restart?

Performance - copying a 11M CSV file character by character

For 99% of cases, you should use buffered input/output operations (we could even say 100% for input).

Use buffered operations unless writing safely is a critical concern.

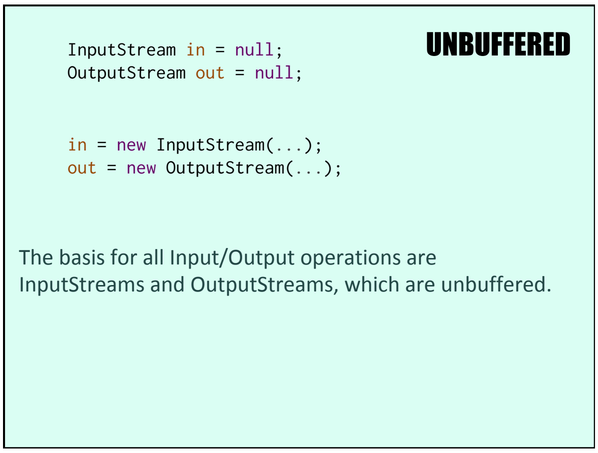
Or for debugging.



Sometimes hard to say what's going on today

Big disk systems have their own, battery protected, buffers (also called CACHE) ← French for Hideout

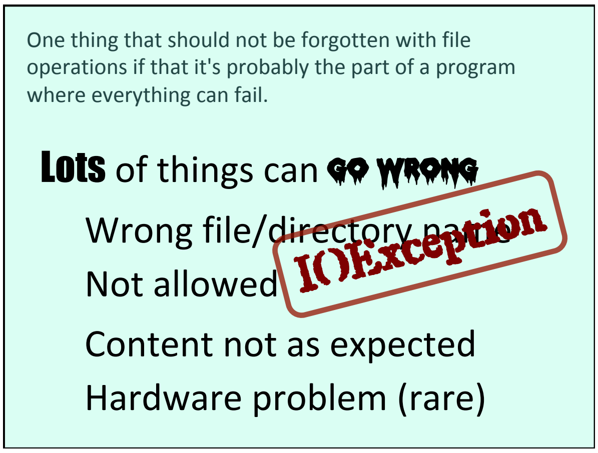
Note that especially with high-end storage you rarely have one level of buffering (in which case unbuffered wouldn't be what it seems). It's a bit hard sometimes to know if the data is on disk or not, and the Cloud doesn't make it any simpler.



UNBUFFERED

```
InputStream in = null;
OutputStream out = null;


in = new InputStream(...);
out = new OutputStream(...);
```

The basis for all Input/Output operations are InputStreams and OutputStreams, which are unbuffered.

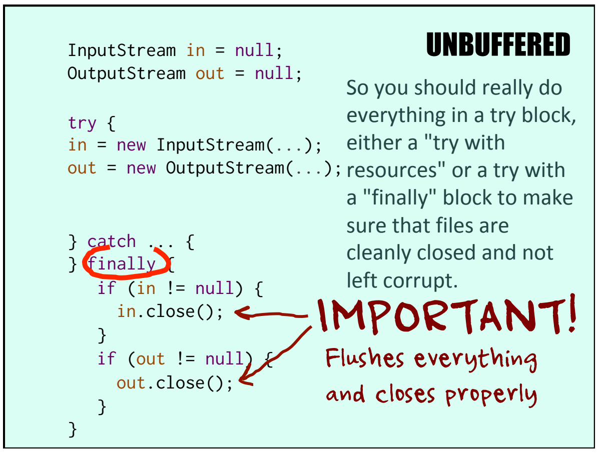


One thing that should not be forgotten with file operations if that it's probably the part of a program where everything can fail.

Lots of things can **GO WRONG**

- Wrong file/directory name
- Not allowed
- Content not as expected
- Hardware problem (rare)

IOException



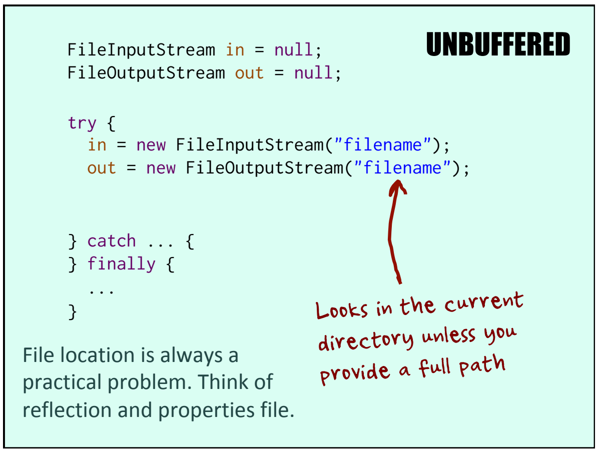
UNBUFFERED

```
InputStream in = null;
OutputStream out = null;

try {
in = new InputStream(...);
out = new OutputStream(...);


} catch ... {
} finally {
  if (in != null) {
    in.close();
  }
  if (out != null) {
    out.close();
  }
}
```

So you should really do everything in a try block, either a "try with resources" or a try with a "finally" block to make sure that files are cleanly closed and not left corrupt.

IMPORTANT!

Flushes everything and closes properly

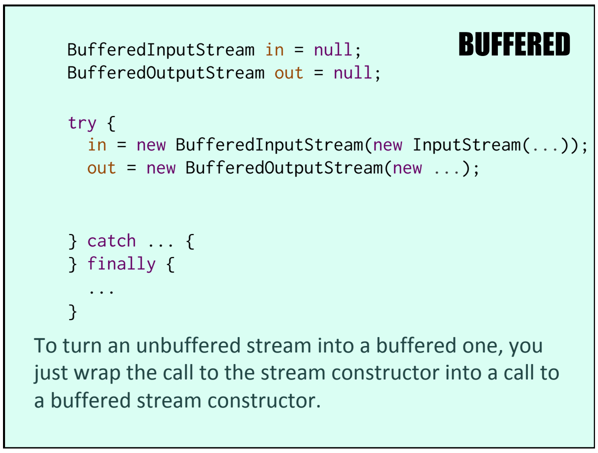


UNBUFFERED

```
FileInputStream in = null;
FileOutputStream out = null;

try {
  in = new FileInputStream("filename");
  out = new FileOutputStream("filename");


} catch ... {
} finally {
  ...
}
```

Looks in the current directory unless you provide a full path

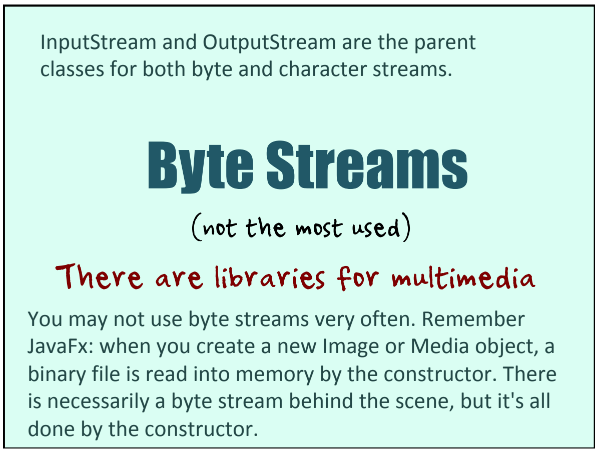
File location is always a practical problem. Think of reflection and properties file.



BUFFERED

```
BufferedInputStream in = null;
BufferedOutputStream out = null;

try {
  in = new BufferedInputStream(new InputStream(...));
  out = new BufferedOutputStream(new ...);


} catch ... {
} finally {
  ...
}
```

To turn an unbuffered stream into a buffered one, you just wrap the call to the stream constructor into a call to a buffered stream constructor.



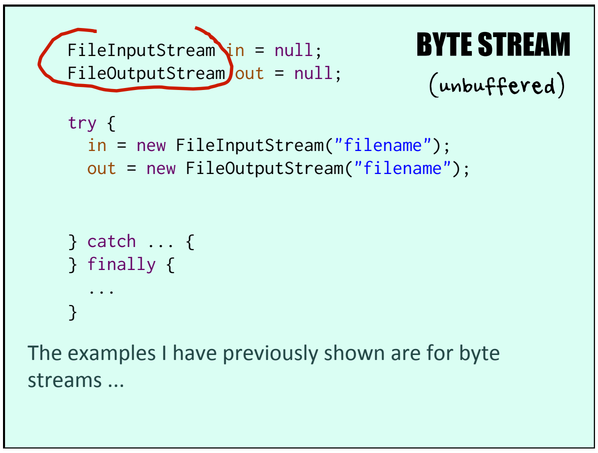
InputStream and OutputStream are the parent classes for both byte and character streams.

Byte Streams

(not the most used)

There are libraries for multimedia

You may not use byte streams very often. Remember JavaFx: when you create a new Image or Media object, a binary file is read into memory by the constructor. There is necessarily a byte stream behind the scene, but it's all done by the constructor.

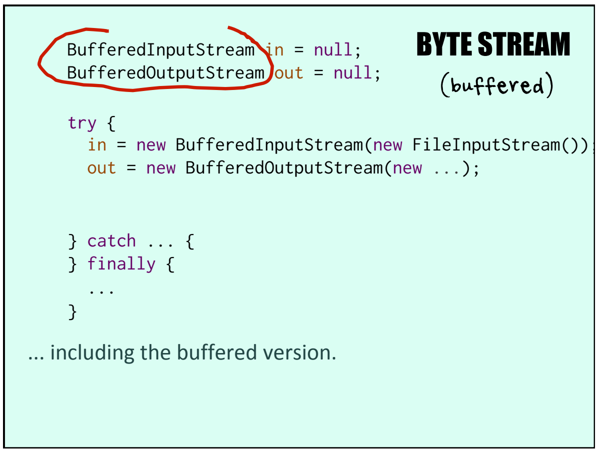


BYTE STREAM

(unbuffered)

```
FileInputStream in = null;
FileOutputStream out = null;

try {
  in = new FileInputStream("filename");
  out = new FileOutputStream("filename");


} catch ... {
} finally {
  ...
}
```

The examples I have previously shown are for byte streams ...



BYTE STREAM

(buffered)

```
BufferedInputStream in = null;
BufferedOutputStream out = null;

try {
  in = new BufferedInputStream(new FileInputStream());
  out = new BufferedOutputStream(new ...);


} catch ... {
} finally {
  ...
}
```

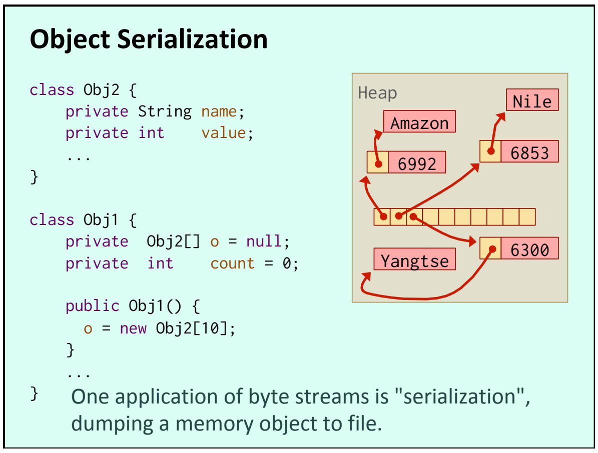
... including the buffered version.

CAUTION

When you talk about "bytes" in input/output operations, you are really dealing with int variables, not byte variables.

HISTORICAL REASON

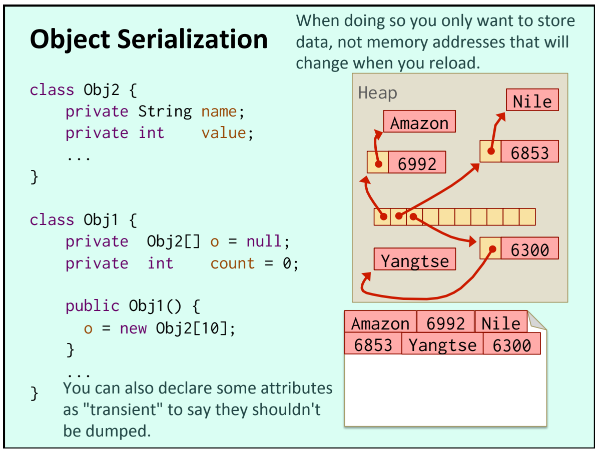
Don't be misled by the "byte" in "byte stream". Operations deal with more than one byte.



Object Serialization

```
class Obj2 {
    private String name;
    private int    value;
    ...
}

class Obj1 {
    private  Obj2[] o = null;
    private  int    count = 0;

    public Obj1() {
      o = new Obj2[10];
    }
    ...
}
```

One application of byte streams is "serialization", dumping a memory object to file.



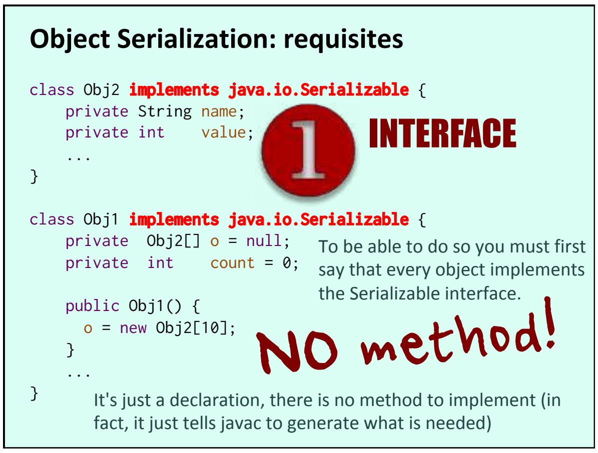
Object Serialization

When doing so you only want to store data, not memory addresses that will change when you reload.

```
class Obj2 {
    private String name;
    private int    value;
    ...
}

class Obj1 {
    private  Obj2[] o = null;
    private  int    count = 0;

    public Obj1() {
      o = new Obj2[10];
    }
    ...
}
```

```
Amazon | 6992 | Nile
6853 | Yangtse | 6300
```

You can also declare some attributes as "transient" to say they shouldn't be dumped.



Object Serialization: requisites

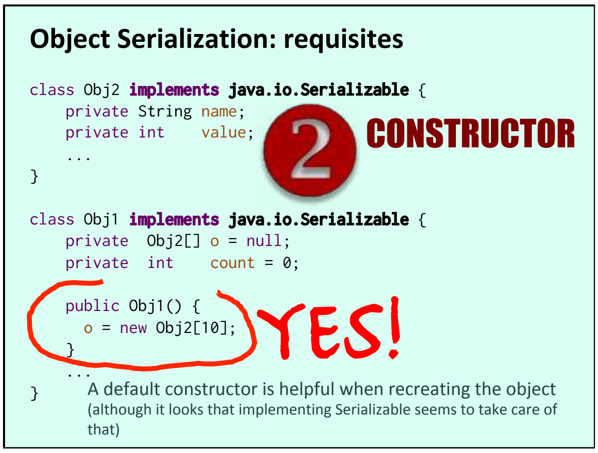
```
class Obj2 implements java.io.Serializable {
    private String name;
    private int    value;
    ...
}

class Obj1 implements java.io.Serializable {
    private  Obj2[] o = null;
    private  int    count = 0;

    public Obj1() {
      o = new Obj2[10];
    }
    ...
}
```

1 INTERFACE

To be able to do so you must first say that every object implements the Serializable interface.

NO method!

It's just a declaration, there is no method to implement (in fact, it just tells javac to generate what is needed)



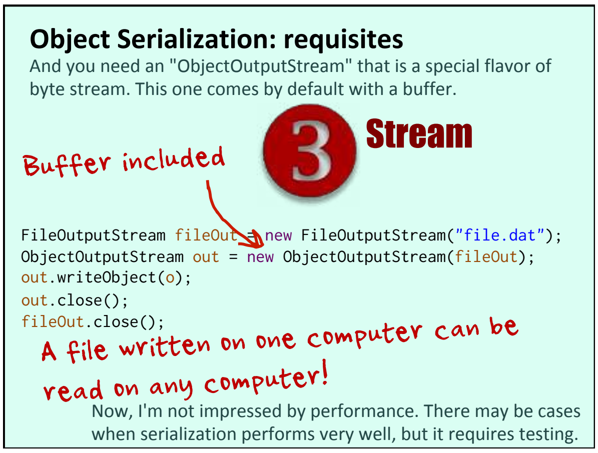
Object Serialization: requisites

```
class Obj2 implements java.io.Serializable {
    private String name;
    private int    value;
    ...
}

class Obj1 implements java.io.Serializable {
    private  Obj2[] o = null;
    private  int    count = 0;

    public Obj1() {
      o = new Obj2[10];
    }
    ...
}
```

2 CONSTRUCTOR

YES!

A default constructor is helpful when recreating the object (although it looks that implementing Serializable seems to take care of that)



Object Serialization: requisites

And you need an "ObjectOutputStream" that is a special flavor of byte stream. This one comes by default with a buffer.

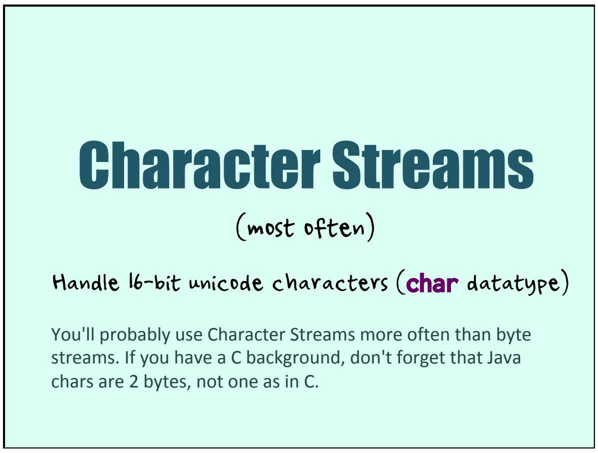
3 Stream

Buffer included

```
FileOutputStream fileOut = new FileOutputStream("file.dat”);
ObjectOutputStream out = new ObjectOutputStream(fileOut);
out.writeObject(o);
out.close();
fileOut.close();
```

A file written on one computer can be read on any computer!

Now, I'm not impressed by performance. There may be cases when serialization performs very well, but it requires testing.

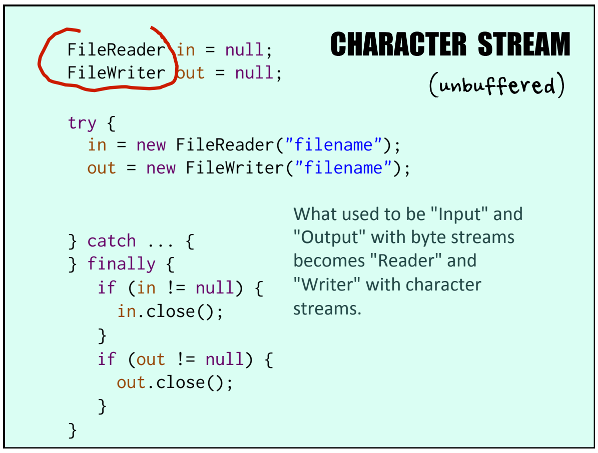


Character Streams

(most often)

Handle 16-bit unicode characters (**char** datatype)

You'll probably use Character Streams more often than byte streams. If you have a C background, don't forget that Java chars are 2 bytes, not one as in C.

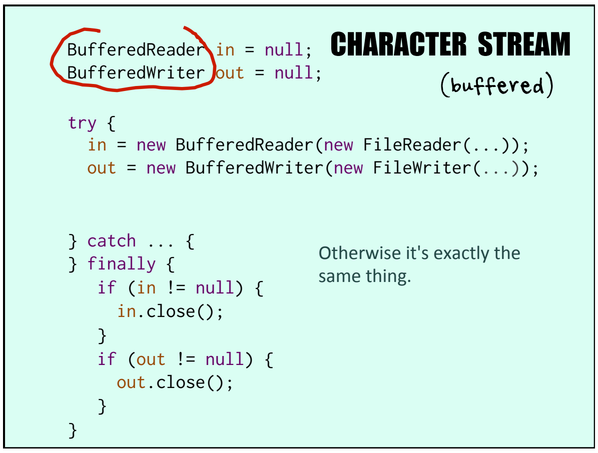


CHARACTER STREAM

(unbuffered)

```java
FileReader in = null;
FileWriter out = null;

try {
  in = new FileReader("filename");
  out = new FileWriter("filename");


} catch ... {
} finally {
   if (in != null) {
     in.close();
   }
   if (out != null) {
     out.close();
   }
}
```

What used to be "Input" and "Output" with byte streams becomes "Reader" and "Writer" with character streams.

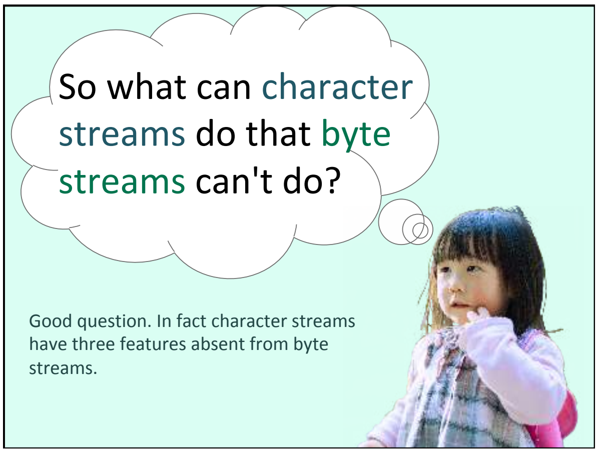


CHARACTER STREAM

(buffered)

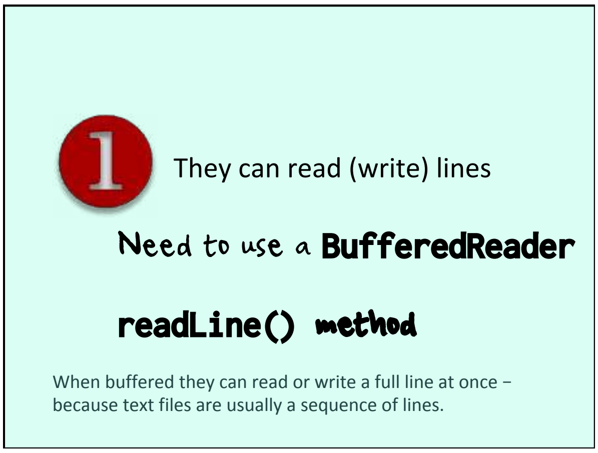
```java
BufferedReader in = null;
BufferedWriter out = null;

try {
  in = new BufferedReader(new FileReader(...));
  out = new BufferedWriter(new FileWriter(...));


} catch ... {
} finally {
   if (in != null) {
     in.close();
   }
   if (out != null) {
     out.close();
   }
}
```

Otherwise it's exactly the same thing.



So what can character streams do that byte streams can't do?

Good question. In fact character streams have three features absent from byte streams.

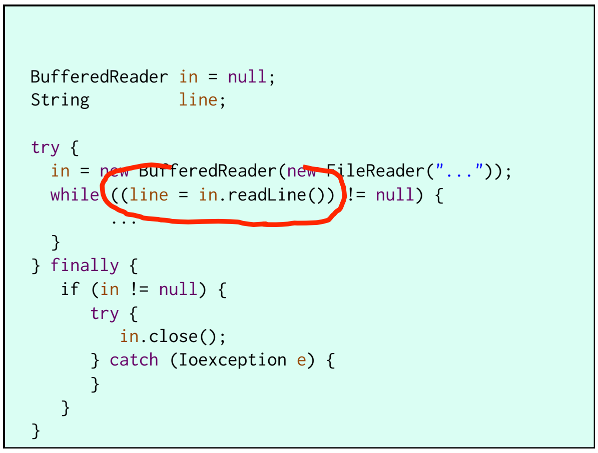


1 They can read (write) lines

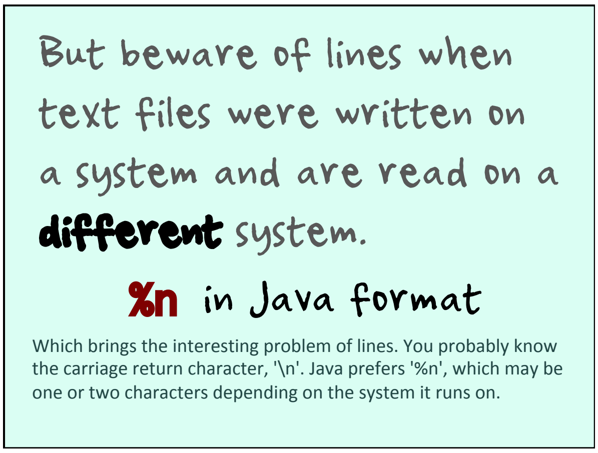
Need to use a **BufferedReader**

readLine() method

When buffered they can read or write a full line at once – because text files are usually a sequence of lines.



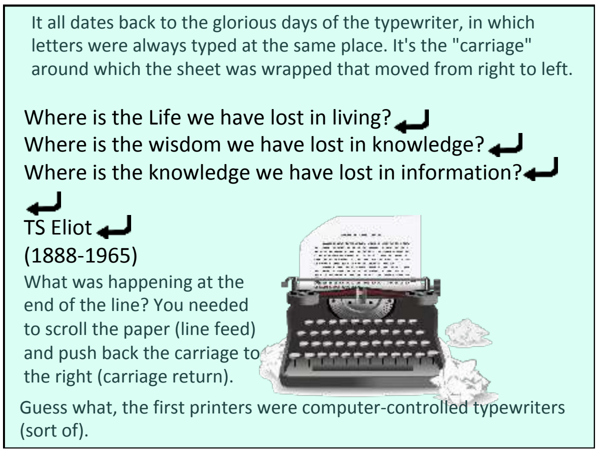
```
BufferedReader in = null;
String         line;

try {
  in = new BufferedReader(new FileReader("..."));
  while ((line = in.readLine()) != null) {
     ...
  }
} finally {
   if (in != null) {
      try {
         in.close();
      } catch (Ioexception e) {
      }
   }
}
```



But beware of lines when text files were written on a system and are read on a **different** system.

%n in Java format

Which brings the interesting problem of lines. You probably know the carriage return character, '\n'. Java prefers '%n', which may be one or two characters depending on the system it runs on.

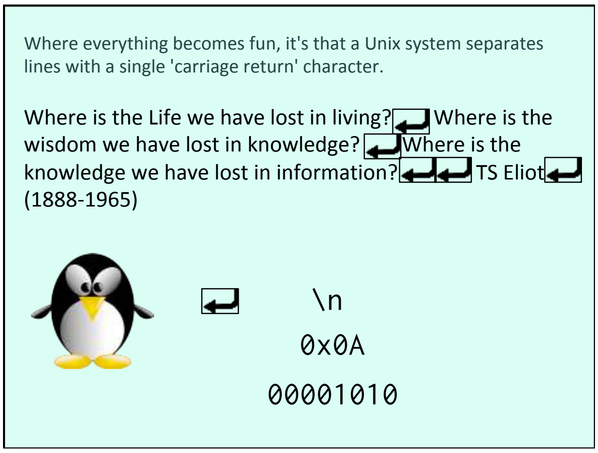


It all dates back to the glorious days of the typewriter, in which letters were always typed at the same place. It's the "carriage" around which the sheet was wrapped that moved from right to left.

Where is the Life we have lost in living?↵
Where is the wisdom we have lost in knowledge?↵
Where is the knowledge we have lost in information?↵
↵
TS Eliot↵
(1888-1965)

What was happening at the end of the line? You needed to scroll the paper (line feed) and push back the carriage to the right (carriage return).

Guess what, the first printers were computer-controlled typewriters (sort of).



Where everything becomes fun, it's that a Unix system separates lines with a single 'carriage return' character.

Where is the Life we have lost in living?↵Where is the wisdom we have lost in knowledge?↵Where is the knowledge we have lost in information?↵↵TS Eliot↵
(1888-1965)

↵ \n

0x0A

00001010

While a Windows system still stores the two separate commands that used to be sent to the printer - line feed (represented by \r), then carriage return.

Where is the Life we have lost in living?↵Where is the wisdom we have lost in knowledge?↵Where is the knowledge we have lost in information?↵↵TS Eliot↵(1888-1965)

↵ \r\n

0x0D0A

0000110100001010

Linux file in a Windows editor

Where is the Life we have lost in living?□ Where is the wisdom we have lost in knowledge?□ Where is the knowledge we have lost in information?□□ TS Eliot□ (1888-1965)

A basic Windows editor (as Linux became more common, many editors became cleverer), looking for \r\n as separator between lines, will see in a Linux text file only one big line with \n characters that it doesn't know how to represent and that it will replace with squares.

Windows file in a Linux editor

Where is the Life we have lost in living?^W
Where is the wisdom we have lost in knowledge?^W
Where is the knowledge we have lost in information?^W
^W
TS Eliot^W
(1888-1965)^W

While on Linux the editor will understand correctly the \n in the Windows file, but will not know what to do with \r and will show it as ^W. There is nothing simple in IT.

There are conversion programs
Many programs understand both.

MOST character files are organized in lines (variable or constant length)

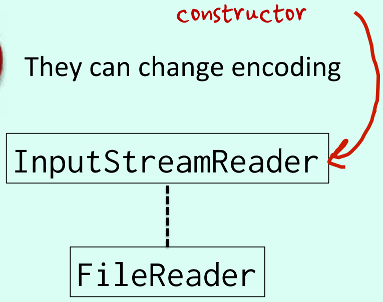
BUT a big character file can sometimes contain a single line (HTML, JSON, XML ...)

You would of course be wrong to believe that a big text file always contains many lines.

Another interesting characteristic is that the parent class of FileReader allows you to specify the character encoding.

You can specify the character set in the constructor

2 They can change encoding



InputStreamReader

FileReader

And finally you can use a Scanner object with a character stream, which is very good at parsing text input.

3 You can use a **Scanner**

Very similar to keyboard and screen, which are character devices

File and Directory Operations

Check the **File** class

Copying, deleting files

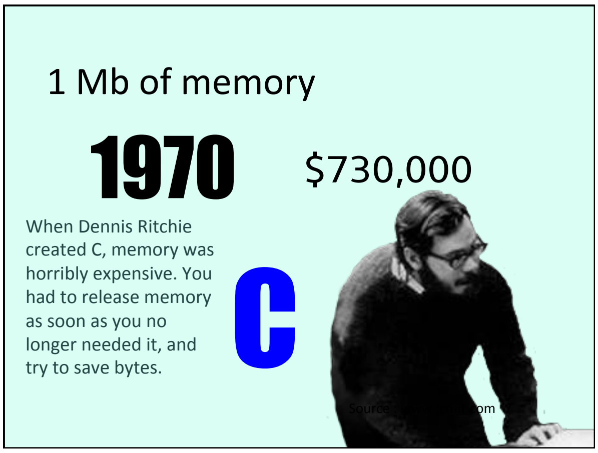
Listing and searching directories

and so forth.

You can do a lot of things with files other than reading and writing them.

Files USED to be very important.

They still are, to some extents, for specialized applications, and they are still the backbone of persistence. However, as an application developers, you are increasingly isolated from files. I have mentioned the case of Image and Media; I could add Properties, and what we'll see the next times, databases that act as a layer between programs and files. A lot of data comes from networks as well.
Every application, 40 years ago, used to open and close a lot of files, this is no longer the case. You open files mostly to load data in memory, and work there. All this is related to the cost of memory.

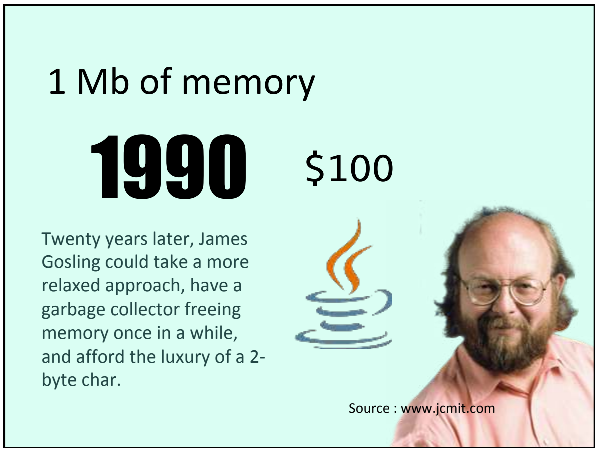


1 Mb of memory

1970 $730,000

When Dennis Ritchie created C, memory was horribly expensive. You had to release memory as soon as you no longer needed it, and try to save bytes.

Source : www.jcmit.com

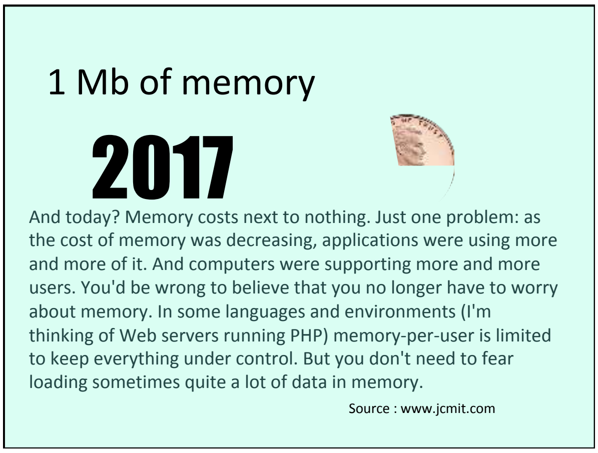


1 Mb of memory

1990 $100

Twenty years later, James Gosling could take a more relaxed approach, have a garbage collector freeing memory once in a while, and afford the luxury of a 2-byte char.

Source : www.jcmit.com



1 Mb of memory

2017

And today? Memory costs next to nothing. Just one problem: as the cost of memory was decreasing, applications were using more and more of it. And computers were supporting more and more users. You'd be wrong to believe that you no longer have to worry about memory. In some languages and environments (I'm thinking of Web servers running PHP) memory-per-user is limited to keep everything under control. But you don't need to fear loading sometimes quite a lot of data in memory.

Source : www.jcmit.com