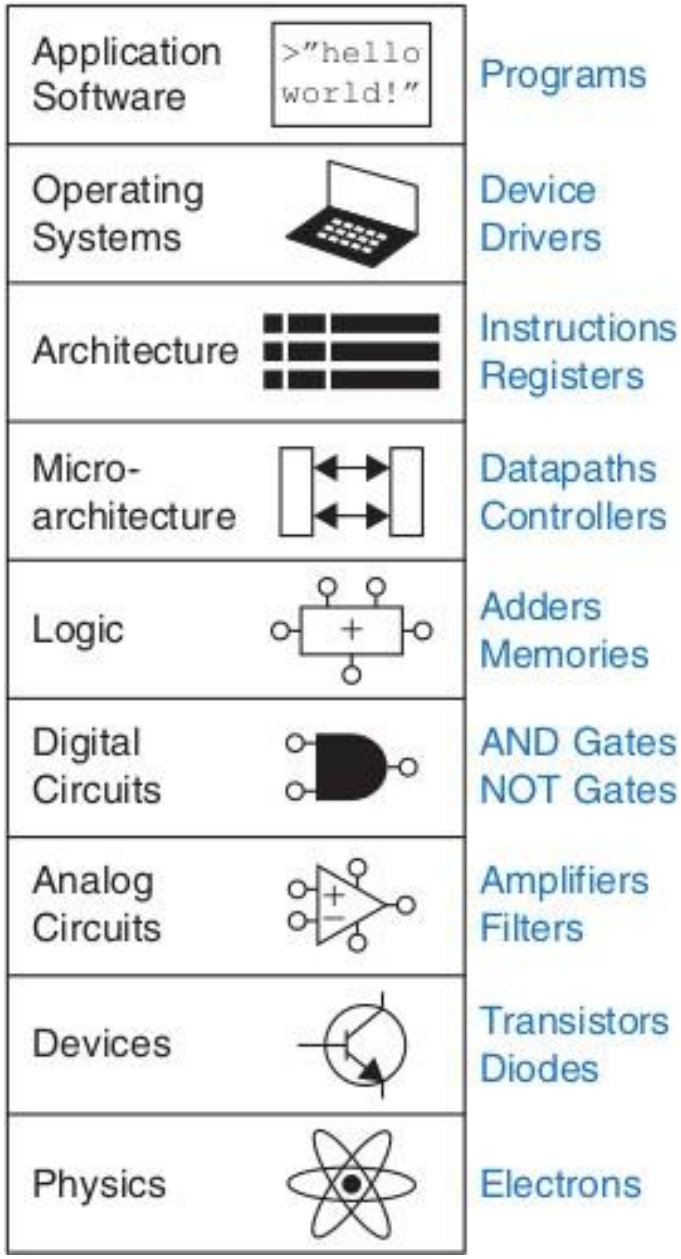
Семинар 6
Codestyle

Abstraction

- **Abstraction** – hiding details when they are not important



by «Digital Design and Computer Architecture»
Harris & Harris

Discipline

- **Discipline** is the act of intentionally restricting your design choices so that you can work more productively

at a higher level of **abstraction**

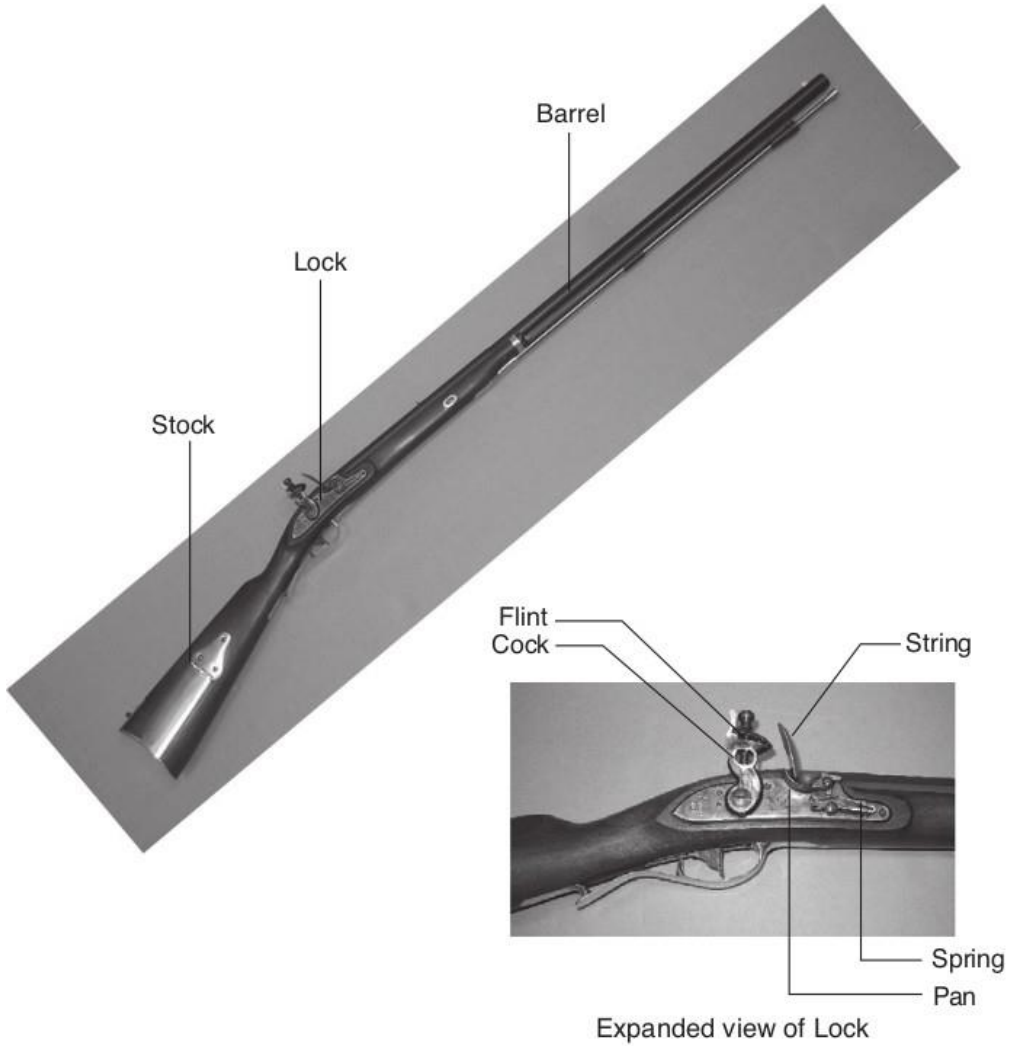


Figure 1.2 Flintlock rifle with a close-up view of the lock
(Image by Euroarms Italia. *www.euroarms.net* © 2006.)

The Three-Y’s

- **Hierarchy** involves dividing a system into modules, then further sub-dividing each of these modules until the pieces are easy to understand.

- **Modularity** states that the modules have well-defined functions and interfaces, so that they connect together easily without unanticipated side effects.

- **Regularity** seeks uniformity among the modules. Common modules are reused many times, reducing the number of distinct modules that must be designed.

Code Style

- Naming:
  - snake_style vs camelStyle
  - bool isPredicate(...) – Which?
  - void push_back(...) – To do what?
  - T size(...) - What?
  - T getLen() / void setLen(T obj)
  - One letter variable, translit variable – very bad
  - xx, yy,… // bad

More Code Style

- Variable declaration – as close as possible to a place of using
- **using namespace** - don’t use! (even using namespace std!)
- Tabs -> 4 spaces
- Too long lines (100 and more characters)
- **Do not copy-paste!**
- Manage whitespaces and blank lines
- Use { } after if / for...
- Don’t use casts. Especially, C-style casts

A bit more Code Style

- Use x += c instead of x = x + c etc.
- Use ++x instead of x++ if possible
- Use ternary operator:
  - if (condition) x = a; else x=b;
  - condition ? x = a : x = b; **BAD**
  - x = (condition ? a : b); **GOOD**

Don’t use magic constants

- Inverse square root implementation, Quake Arena III

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point
bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this
can be removed

    return y;
}
```

Tips, tricks && best practices

- Don’t compare floats/doubles like var_1 == var2
  - abs(var_1 - var_2) < eps (~1e-5)
- Use **const** and **references** in function arguments if possible
- Make difference between **struct** and **class**