- The 1960s gave birth to **structured programming.**
- Languages such as C and Pascal.
- Structured languages are characterized by their support for stand-alone subroutines, local variables, rich control constructs, and their lack of reliance upon the GOTO
- ***process-oriented model***
  - *code acting on data*

- However, problems with this approach appear as programs grow larger and more complex.
- Not very useful for large projects

  =>Use **Object Oriented Programming (1980)**
- ***Object-oriented programming***
  - *data controlling access to code*
  - *OO languages such as C++, JAVA*

- Object-oriented programming
  - provides facilities or mechanisms that allow use of the OO paradigm
  - provides compile time and/or run time checks against unintentional deviation from the paradigm

- A language is **object-oriented** if and only if it satisfies the following (Cardelli and Wegner):
  - It supports objects that are data abstractions with an interface of named operations and a hidden local state.
  - Objects have an associated type [class].
  - Types [classes] may inherit attributes from super-types
- A language is considered **object based** if it directly supports data abstraction and classes

Abstraction

- Humans manage complexity through abstraction.
- For example, people do not think of a car as a set of tens of thousands of individual parts
- They think of it as a well-defined object with its own unique behavior
- A good abstraction is achieved by having
  - meaningful name reflecting the function
  - **minimum** and at the same time **complete** features
  - coherent features (relatedness)

- Central to the idea of an abstraction is the **concept of invariance.**
- An *invariant is* some Boolean condition whose truth must be preserved.
- For each operation associated with an object, we may define
  - *preconditions (invariants* assumed by the operation) as well as
  - *postconditions (invariants satisfied by the* operation)
- Violating an invariant breaks the contract associated with an abstraction.

```
int max(int n, const int a[]) {
int m = a[0];
int i = 1;
while (i != n) {
// m equals the maximum value in a[0...i-1]
     if (m < a[i]) m = a[i];
// m equals the maximum value in a[0...i]
     ++i;
// m equals the maximum value in a[0...i-1]
}
// m equals the maximum value in a[0...i-1],
//and i==n
return m;
}
```

```
#include <assert.h>
int test_assert(int x, int m, const int a[])
{
    for(i = 0; i < x; i++) assert(a[i] <= m);
}
int max(int n, const int a[]) {
     int m = a[0];
     int i = 1;
     while (i != n) {
          test_assert(i, m, a);
          if (m < a[i]) m = a[i];
          ++i;
     }
```

- Loop invariant (noun) versus loop-invariant (adjective) code

```
for (int i=0; i<n; ++i) {
        x = y+z; a[i] = 6*i + x*x;
}
```

```
x = y+z;
t1 = x*x;
for (int i=0; i<n; ++i) {
        a[i] = 6*i + t1;
}
```

- loop-invariant code motion

- If a precondition is violated, this means that a client has not satisfied its part of the bargain
- Similarly, if a postcondition is violated, this means that a server has not carried out its part of the contract
- An exception is an indication that some invariant has not been or cannot be satisfied.
- Certain languages permit objects to throw exceptions

Design By Contract

- All abstractions have **static** as well as **dynamic** properties.
- A file object takes up a certain amount of space in memory; it has a name, and it has contents. These are all static properties.
- The value of each of these properties is dynamic, relative to the lifetime of the object:
  - A file object may grow or shrink in size,
  - Its name may change,
  - its contents may change.

- Procedure-oriented programming
  - The activity that changes the dynamic value of objects is the central part of all programs
- Object-oriented programming
  - Things happen whenever we send a message to an object
- What operations we can meaningfully perform on an object and how that object reacts constitute the entire behavior of the object.