2.1 Objects
An object is a particular instance of a class and there may be multiple instances of a class. An object has

- data, called attributes, fields or data members, and
- functions, called methods or member functions.

A member function is called with the . notation. For example:

```
object.method();
```

The code

```
MyString puppet;
```

during compilation declares variable puppet to be an object of class MyString, and during execution causes the program to create the variable puppet by reserving memory for it and initializing it by executing the no-argument constructor. (See constructors below.)

2.2 Data Members and Member Functions
Every member function can access every data member. But, usually all data and some member functions are labeled private; user methods are labeled public. (There are other possibilities.) Private means that the external user of this class cannot see or use it.

Member functions execute on an object of that class. Most classes have two types of member functions:

- accessor functions: these allow the user to get data from the object.
- mutator functions: these allow the user to set data in the object.

2.3 Constructors
A constructor is a special function that initializes the state of the object; it has the same name as the class, but does not have a return type. There can be more than one constructor. Note that the compiler will provide a default no-argument constructor if none is coded. Some constructor is always executed when an object is created.
2.4 Why Objects?

Object-oriented programming rests on the three basic principles of *encapsulation*:

- **Abstraction**: ignore the details
- **Modularization**: break into pieces
- **Information hiding**: separate the implementation and the function

OOP uses the idea of classes. A **class** is a structure which houses data together with operations that act on that data. We strive for **loose coupling**: each class is largely independent and communicates with other classes via a small well-defined interface. We strive for **cohesion**: each class performs one and only one task (for readability, reuse).

We strive for **responsibility-driven design**: each class should be responsible for its own data. You should ask yourself: What does the class need to know? What does it do?

The power of OOP also comes from two further principles which we will discuss later:

- **Inheritance**: classes inherit properties from other classes (which allows partial code reuse)
- **Polymorphism**: there are multiple implementations of methods and the correct one is executed

**Sample Code**

Below is a sample class and a main function. But note that there are several style problems with it, some of which we will fix later. The output is

GI Joe can drink
Barbie can’t drink

\[\text{Citizen.cpp}\]