Classes/Encapsulation (+ constructors, abstraction, introduction of public and private, stuct vs class)

Course objectives covered: *constructors, encapsulation, abstraction, struct vs class, public and private members of our class*

Last lecture, we talked about the idea of encapsulating properties of objects into a class.

Encapsulation-bundling up info/actions into a container-in C++, a class **(also see Stroustrup's glossary for his definition).** We can then make objects from this class (the class acts as a blueprint for any objects we make).

The book gives the following brief summary of a class: (16.2, pg 450)

1. A class is a user defined type
2. A class consists of a set of members
    a. The most common are data members and member functions
3. Members are accessed using . (dot) for objects and -> (arrow) for pointers
4. Operators can be defined for a class
5. **A struct is a class where members are by default by public (a class has all members private by default)**

Abstraction: Specifying a general interface while hiding implementation details **(also see Stroustrup's glossary for his definition-notice it intertwines encapsulation, abstraction and information hiding).**

Imagine you have a clock:

Our only goals for our clock are:

1. Have the alarm ring at a certain time
2. Have the clock let us know what time it is

The actual inner mechanics of the clock are of no interest to us. They still exist of course, but we do not care how it works-as long as our goals are achieved (above), we're fine.

When we interact with our clock:

- we only want it to display the time and ring when the alarm is set.
- The actual functioning of the clock (like the cogs working) is hidden from us (even though it is obviously necessary and occurring).
- Classes allow us to implement abstraction in C++ by:
    - containing all the characteristics and functionality of our object (like a clock)
    - making certain characteristics and functionality of it visible using keywords like **public** and **private**
        - We can hide our information from the outside word so the outside world does not need to worry about implementation details-they can just interact with the object as expected

Before we continue:

1) I want to mention that someone who is interested in the mechanics of a clock could have some sort of abstraction by not caring how each piece is shaped and cut out of the metal-only that it is the right shape etc. Abstraction is like a viewpoint on your object (what is visible and what is not at any given viewpoint) and depending on what you are trying to do (a consumer using a clock vs someone fixing a clock), you will have different viewpoints.

2) Another abstraction example: a TV. We interact with the TV pushing buttons or using the remote-all the inner workings don't matter to us (even though they are necessary for the TV to function). A person fixing a TV however would care about these inner workings (so a different abstraction view), but would most likely not care where or how each piece was sourced (unless price is factor, for example).

Constructors:

- You can think of a constructor as a way to initialize your object
    - If you want to create a Puppy object from a Puppy class, you can create it as a specific breed or color (instead of making a generic Puppy object, then modifying the attribute of breed as a second step)
    - A special class member that creates and initializes an object from the class
    - Also see Stroustrup's glossary for his definition

- Note that you could also have created a function (called initialize() for example) that does this (but a constructor is a more natural way to create an object)

- You can have an overloaded constructor (meaning you have different ways to initialize your object)

- When you do not create a user-defined constructor for your class, a default constructor is used
    - Meaning there are no parameters OR
    - There are default parameters (we will do an example of this in a future lecture)

Constructor Example:

Professor Goody is studying monkeys for her research and wants to enter every monkey she sees into a program for further study. She is specifically trying to document if they look happy or not.

She sees monkeys in different scenarios:

1. The monkey has a certain number of bananas and not expressing an emotion (but she knows if the monkey has less than 40 bananas it is not happy)
2. The monkey has a certain number of bananas and is clearly expressing an emotion
3. A monkey with no visible number of bananas and no visible emotion

Sample Run:

```
computer$ g++ -o monkey monkey.cpp
computer$ ./monkey
Monkey status: HAPPY!!! with 370 banana(s)!
Monkey status: NOT HAPPY :( with 33 banana(s).
??? Monkey status and number of bananas currently unknown... ???
```

```
#include <iostream>
#include <string>

using namespace std;

class Monkey{
//default in a class is private (since I am not specifying public or private here, it is private)
  int num_bananas;
  bool happy;

//since this member function is private, the outside world can't access it. In this example, there
//is no need for the outside world to access it since it just an implementation detail used in the
//constructor to show the money status
  void print_status()
  {
     if(happy)
     {
        cout << "Monkey status: HAPPY!!! with " << num_bananas << " banana(s)!"<<endl;
     }

     else
     {
        cout << "Monkey status: NOT HAPPY :( with " << num_bananas << " banana(s)."<<endl;
     }
  }

  public:
    Monkey(int banana)
    {
       num_bananas=banana;
```

```
      if(banana<40)
      {
       happy=false;
      }
      else
      {
       happy=true;
      }
      print_status(); //we can use our private function here since we are still in the class
   }

   Monkey(int banana, string feelz)
   {
      num_bananas=banana;

      if(feelz=="happy")
      {
         happy=true;
      }

      else
      {
         happy=false;
      }
      print_status();
   }

    Monkey()
    {
        cout << "??? Monkey status and number of bananas currently unknown... ???" <<endl;
    }
};

int main()
{
    Monkey m1(370);
    //m1.print_status(); //can't do cuz private
    Monkey m2(33, "bored");
    Monkey m3;
}
```

Course objectives covered: *sort function, check if vector is empty (using empty function), operator [] vs at() to see value (using at in this code-at is more c++ style), getter functions (accessor methods) to access private members*

Program 1:

The athletic department has asked you to create a program that allows athletes to log their times (in seconds) and decide whether or not to make times visible. If times are visible, then only the best or worst time can be seen.

```
computer$ g++ athlete.cpp
computer$ ./a.out
First athlete's name: Pat
Enter the second athlete's name: Scotty
Scotty, can we share your times?
no
-Ok, times will be made private.

Who is logging a time?
Scotty
Enter new time (in seconds):
34
No times to show OR user has made times private.

Who is logging a time?
Scotty
Enter new time (in seconds):
99
No times to show OR user has made times private.

Who is logging a time?
Pat
Enter new time (in seconds):
15
Would you like to see the best time or worst time?
neither
Times will not be shown.

Who is logging a time?
Pat
Enter new time (in seconds):
11
Would you like to see the best time or worst time?
worst
Worst time is: 15

Who is logging a time?
Pat
Enter new time (in seconds):
5
Would you like to see the best time or worst time?
best
Best time is: 5

Who is logging a time?
exit
Exiting...
```

```
#include <iostream>
#include <vector>
#include <string>

class Athlete{

  //by default, these are private
   std::string name;
   std::vector <int> time_seconds; //times in seconds, ordered lowest to highest

 public:
   bool no_show; //can we share times? true yes, false no

   Athlete(std::string name, bool no_show) //constructor 1
   {
```

```
    this->name=name;
    this->no_show=no_show;
  }

  Athlete(std::string name) //constructor 2
  {
    this->name=name;
    this->no_show=true; //default is yes you can show times
  }

  void log_time()
  {
    int answer;
    std::cout<<"Enter new time (in seconds): "<<std::endl;
    std::cin>>answer;

    time_seconds.push_back(answer);
    std::sort(time_seconds.begin(), time_seconds.end()); //put in order of best to worst using sort()
  }

  void check_time()
  {
     std::string answer;

    //we can use size or empty to see if vector is empty
    if(time_seconds.empty()|| !no_show) //if there are no times logged or the athlete has their times set
to private
    {
      std::cout<<"No times to show OR user has made times private."<<std::endl;
    }

    else //can show (but only shows highest or lowest)
    {
      std::cout<<"Would you like to see the best time or worst time?"<<std::endl;
      std::cin>>answer;

     if(answer=="best")
     {
      std::cout<<"Best time is: "<<time_seconds.at(0)<<std::endl; //using at (we could also use []
operator)
     }

     else if(answer=="worst")
     {
       std::cout<<"Worst time is: "<<time_seconds.at(time_seconds.size()-1)<<std::endl;
     }

     else
    {
      std::cout<<"Times will not be shown."<<std::endl;
     }
```

```
    }
  }

  std::string get_name() //getter function (since name is private)
  {
    return name;
  }
};

int main(int argc, char **argv)
{
  std::string answer;
  //creating two athlete objects
   Athlete a("Pat"); //default is true to show times
  std::cout<<"First athlete's name: "<<a.get_name()<<std::endl;

  std::cout<<"Enter the second athlete's name: ";
  std::cin>>answer;

 Athlete b(answer); //default is true to show times
  std::cout<<answer<<", can we share your times?"<<std::endl;
  std::cin>>answer;

  if(answer=="yes")
  {
     //No need already set as true when Athlete object was created
     std::cout<<"-Ok, times will be made public.\n"<<std::endl;
  }

  else
  {
    b.no_show=false; //We can directly access no_show since it is public in the class
    std::cout<<"-Ok, times will be made private."<<std::endl;
  }

  while(answer!="exit")
  {
    std::cout<<"\nWho is logging a time?"<<std::endl;
    std::cin>>answer;

    if(answer=="Pat") //athlete 1-we hardcoded this
    {
      a.log_time();
      a.check_time();
    }

    else if(answer==b.get_name()) //using a getter function (since name is private)-name is what user
entered
    {
      b.log_time();
      b.check_time();
```

```
    }

    else if(answer=="exit")
    {
      std::cout<<"Exiting..."<<std::endl;
    }

    else //unknown response
    {
      std::cout<<"No athlete with this name."<<std::endl;
    }
  }
}
```