Member Functions

Due this week

- **HW 6**
  - Write solutions in VSCode and paste in **CodeRunner**.
  - Write assertions in main()
  - Extra-credit – start early
  - Zip your .cpp files and submit on canvas.
- Check the due date! **No late submissions!!**
- **Quiz 6.** Check the due date! **No late submissions!!**

Practicum 2 – 100 points

- Coming up in week 9: October 17th @ 9:05 am
- Covers material from weeks 1 – 6 (exceptions posted on Canvas)
  - Variables
  - Conditionals
  - Strings
  - Loops
  - Functions
  - Arrays
- Practice Questions also on Canvas

Class

- A class describes a set of objects with the same behavior
- Variables of a class are called objects

- Every class has:
  - Data members
  - Member functions

Example

We have used the string class, but we didn’t have to deal with how str.substr(6) works, or what str[6] is actually doing.

- We had access to the **public interface** to the string class, and just got to use that
- Protects the class from us accidentally messing it up

A generic class interface

```
class NameOfClass
{
  public:
    // the public interface

  private:
    // the data members
};
```

Use CamelCase for the names of classes

Any part of our program should be able to call the member functions.
→ they go in the public interface

Data members are defined in the *private section* of the class. Only member functions (within our class) can access the data members. They're hidden from the rest of the program
→ they go in the private section of the class

Designing a class: pokemon

```
class Pokemon
{
public:
    // function prototypes
private:
    string name_;
    int hit_points_;
};
```

Member Functions

Two types:

1. Mutators / setters
2. Accessors / getters

Designing a class: pokemon

```
class Pokemon
{
public:
    void setName(string n);
    void setHP(int h);
    string getName() const;
    int getHP() const;
private:
    string name_;
    int hit_points_;
};
```

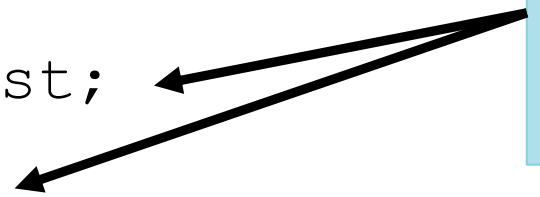
These are function prototypes.
We'll define them later.

What is const?

```
class Pokemon
{
public:
    void setName(string n);
    void setHP(int h);
    string getName() const;
    int getHP() const;
private:
    string name_;
    int hit_points_;
};
```

What is const?

```
class Pokemon
{
public:
    void setName(string n);
    void setHP(int h);
    string getName() const;
    int getHP() const;
private:
    string name_;
    int hit_points_;
};
```



getters only report the values of data members, and never alter them
→ we declare these functions to be const so they can't mess our stuff up

Dot Notation

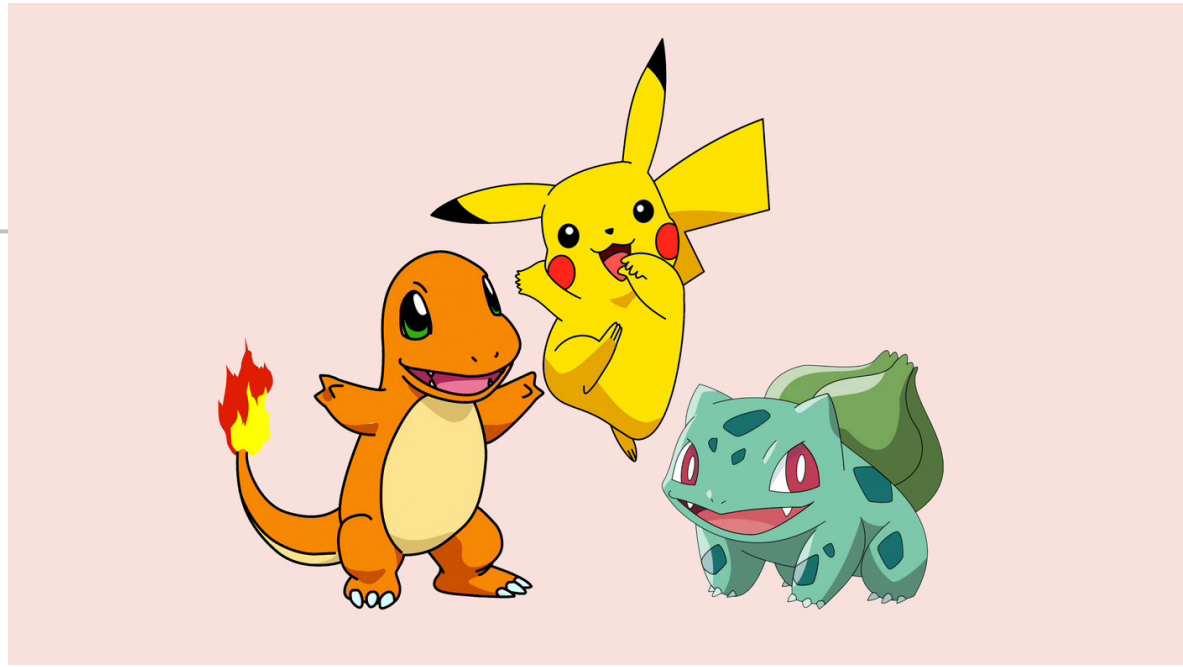
You call the member functions by first creating a variable of type **Pokemon** and then using the dot notation:

```
Pokemon pikachu;
...
pikachu.setName("pikachu");
pikachu.setHP(80);
...
int health = pikachu.getHP();
cout << "Pikachu hp: " << health << endl;
```

Encapsulation

Every `Pokemon` object has its own copy of these data members

```
Pokemon pikachu;
Pokemon charmander;
… [use setter functions] ...
```



pikachu

```
name_ = pikachu
hit_points_ = 100
level_ = 1
type_ = electric
```

charmander

```
name_ = charmander
hit_points_ = 70
level_ = 1
type_ = fire
```

…..

Encapsulation

The private data members are only accessible via member functions:

- Won't work: `Pokemon pikachu;`

```
… [use setter functions] ...
cout << pikachu.name_ << endl;
```

Encapsulation

The private data members are only accessible via member functions:

- Won't work:
```
Pokemon pikachu;
… [use setter functions] ...
cout << pikachu.name_ << endl;
```
- Will work!
```
Pokemon pikachu;
… [use setter functions] ...
cout << pikachu.getName() << endl;
```

Encapsulation

- You can move data members to the public interface and make it accessible
- DON’T! It is not good practice
  - Will keep things tidier and easy to debug!

Encapsulation

- We might want to change how data members are computed and/or manipulated, but the important details (data members) shouldn't necessarily change.
- Example:
  - We can write the mutator for `level_` so it can never be negative
  - On the other hand, if `level_` were public, we could just straight up set it to be negative.

The Interface

- The interface should not change even if the details of how they are implemented change.
- A driver switching to an electric car does not need to re-learn how to drive.

Class Implementation

Class Implementation

```
class Pokemon
{
public:
    void setName(string n);
    void setHP(int h);
    string getName() const;
    int getHP() const;
private:
    string name_;
    int hit_points_;
};
```

Now that we have the interface, we need to actually define the prototypes!
→ start by **implementing the member functions**

Implementing member functions

- Start with the setName() member function:

```
void setName( string name ) {
      name_ = name;
}
```

- One more thing to add: as written, there is no connection to the Pokemon class!

Implementing member functions

- Start with the setName() member function:

```
void setName( string name ) {
      name_ = name;
}
```

- One more thing to add: as written, there is no connection to the Pokemon class!
- so we specify for our member functions:

```
Pokemon::[member function name]
```

Implementing member functions

- We do not need the Pokemon:: declaration when defining the class:

```
class Pokemon {
      public:
      ...
      void setName( string name );
      …
      private:
      ...
};                                    // no need to add ;
void Pokemon::setName( string name )  // we are defining the function here
{
      name_ = name;
}
```

Constructors

- A constructor is a member function that initializes the data members of an object.
- The constructor is automatically called whenever an object is created.

```
Pokemon pikachu;
```

- (You don’t see the function call nor the definition in the class, it but it’s there.)

Motivation

- By supplying a constructor, by writing our own implementation, you can ensure that all data members are properly set before any member functions act on an object.
- To understand the importance of constructors, consider:

```
Pokemon pikachu;
pikachu.setName("pikachu");
int health = pikachu.getHP(); // May not be 1
```

- Notice that the programmer forgot to call **set initial values** before calling getters.

Constructor Code

- You declare constructor functions in the class definition. There must be **no** return type, not even **void**.
- The name of the constructor must be the same as the class:

```
class Pokemon
{
public:
   Pokemon(); // A constructor
...
};
```

- The constructor definition resembles other member functions:

```
Pokemon:: Pokemon()
{
   hit_points_ = 100;
   level_ = 1;
}
```

Default Constructors

- If you do not write a constructor for your class, the compiler automatically generates one for you, which does nothing but allocate memory space for the data members.

- The compiler does NOT provide safe initial data values, EXCEPT that `string` members are initialized to “”.

- Default constructors are called when you define an object and do not specify any parameters for the construction.

```
Pokemon pikachu;
```

Parameterized Constructors

- Constructors can have parameters, and can be overloaded :

```
class Pokemon
{
public:
  // “Default” constructor: Sets hp & evolution = 0
  Pokemon();
  // Sets name_ = n, type_ = t, hit_points_ = h and level_ = e
  Pokemon(string n, int h, int e, string t);
private:
  int hit_points_;
  int level_;
  string name_;
  string type_;
};
```

Overloaded Constructors

- When the same name is used for more than one function, then the functions are called **overloaded**. The compiler determines which to use, based on the parameter list of the call.
- When you construct an object, the compiler chooses the constructor that matches the parameters that you supply:

```
Pokemon(); // Uses default constructor

Pokemon(“pikachu”, 100, 1, “electric”); // uses
                                        parameterized constructor
```

Common Error: Resetting objects

- You cannot call a constructor with dot notation to “reset” an object.

```
Pokemon pikachu;
...
pikachu.Pokemon(); // Syntax Error
```

- The correct way to reset an object is to construct a new one and assign it to the old:

```
pikachu = Pokemon(); //creates an
// unnamed object, then copies it to pikachu
```