Destructors

In this lesson, we will study the purpose of destructors.

WE'LL COVER THE FOLLOWING

What is a Destructor?

A **destructor** is the opposite of a constructor. It is called when the object of a class is **no longer in use**. The object is destroyed and the memory it occupied is now free for future use.

`C++` does not have transparent garbage collection like `Java`. Hence, in order to efficiently free memory, we must specify our own destructor for a class.

In this destructor, we can specify the additional operations which need to be performed when a class object is deleted.

A class destructor can be created in a similar way to the constructor, except that the destructor is preceded by the `~` keyword.

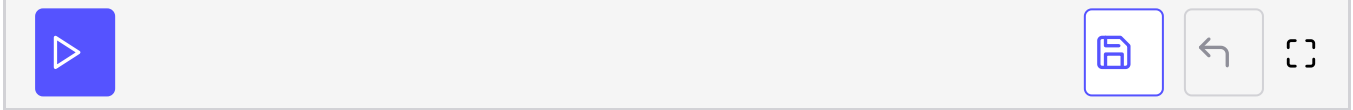
Explicit Garbage Collection

A small degree of garbage collection can be easily achieved through smart pointers. A smart pointer, the `shared_ptr` in particular, keeps a reference count for the object it points. When the counter comes down to `0`, the object is deleted.

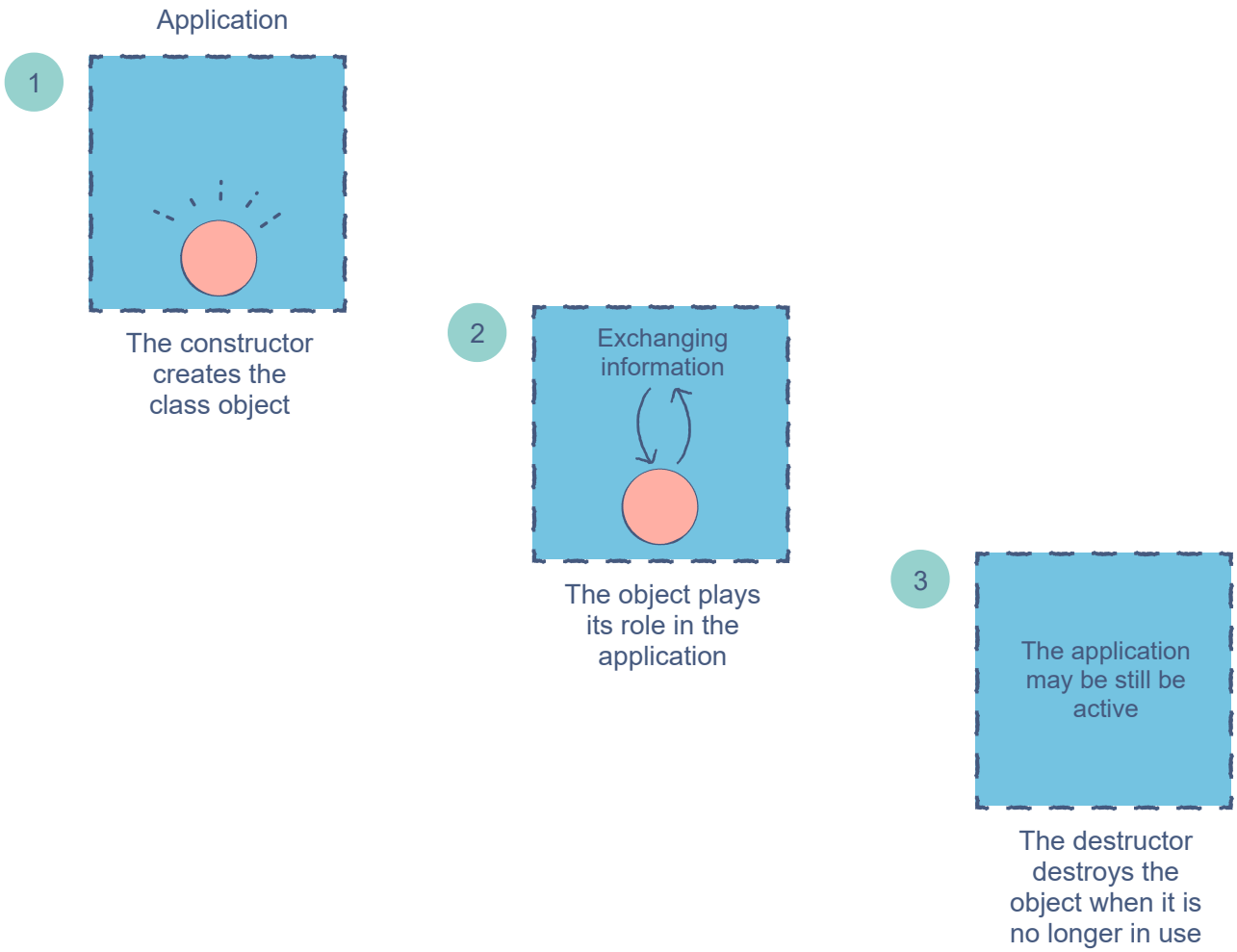
It's time to make a destructor and see how it behaves.

Example

```
#include <iostream>
#include <string>
using namespace std;

class Date {
  int day;
  int month;
  int year;

  public:
  // Default constructor
  Date(){
    // We must define the default values for day, month, and year
    day = 0;
    month = 0;
    year = 0;
  }

  // Parameterized constructor
  Date(int d, int m, int y){
    // The arguments are used as values
    day = d;
    month = m;
    year = y;
  }

  // A simple print function
  void printDate(){
    cout << "Date: " << day << "/" << month << "/" << year << endl;
  }

  ~Date(){
    cout << "Deleting date object" << endl;
  }
};

int main(){
  Date d(1, 8, 2018);
  d.printDate();
}
```



As we can see, the destructor is automatically called and the memory is freed up. What's interesting is that the `cout` statement we specified is also executed on the destructor call.



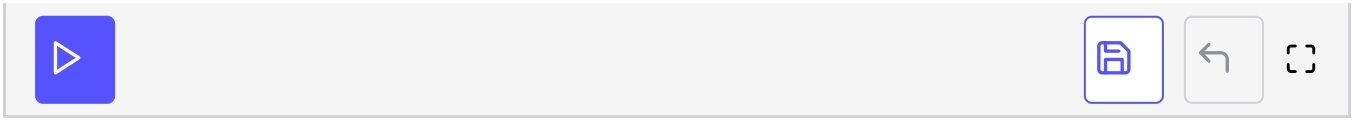
The lifecycle of an object.

Destructors and Pointers

In the case of pointers, destructors are called when we issue the `delete` command:

```
#include <iostream>
#include <string>
using namespace std;

class Date {
  int day;
  int month;
  int year;

  public:
  // Default constructor
  Date(){
    // We must define the default values for day, month, and year
    day = 0;
    month = 0;
    year = 0;
  }

  // Parameterized constructor
```

```
  Date(int d, int m, int y){
    // The arguments are used as values
    day = d;

    month = m;
    year = y;
  }

  // A simple print function
  void printDate(){
    cout << "Date: " << day << "/" << month << "/" << year << endl;
  }

  ~Date(){
    cout << "Deleting date object" << endl;
  }
};

int main(){
  Date* d = new Date(1, 8, 2018); // Object created in dynamic memory
  d->printDate();
  delete d;
  cout << "End of program" << endl;
}
```



Destroying Objects is Important

If we don't deallocate the memory for the objects which are not in use, we will end up with **memory leaks** and no space for our application to work long term.

In the next lesson, we will learn about friend functions and their uses.