Constructor Rules for Inheritance in Derived Classes:

The 'default constructor' and the 'destructor' of the "base class" are always called when a "new object"
of a derived class is created or destroyed.

Example:

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

// Base class
class Base {
public:
    Base() {
        cout << "Base constructor called" << endl;
    }

    ~Base() {
        cout << "Base destructor called" << endl;
    }
};

// Derived class inheriting from Base
class Derived : public Base {
public:
    Derived() {
        cout << "Derived constructor called" << endl;
    }

    ~Derived() {
        cout << "Derived destructor called" << endl;
    }
};

int main() {
    cout << "Creating a Derived object:" << endl;
    Derived derivedObj; // Creating an object of the Derived class

    cout << "\nDestroying the Derived object:" << endl;
    // The destructor of Derived will be called first, then the destructor of Base
    return 0;
}
```

Output:

```
Creating a Derived object:
Base constructor called
Derived constructor called

Destroying the Derived object:
```

```
Derived destructor called
Base destructor called
```

-------------------------------

--> Constructor Rules for Derived Classes:
                                          You can also specify an constructor of the
base class other
                                          than the default constructor

You can specify a constructor for the base class other than the default constructor by explicitly calling that constructor in the initialization list of the derived class constructor. This allows you to pass arguments to the base class constructor when creating objects of the derived class.

SyntaX:

```
             DerivedClassCon ( derivedClass args ) : BaseClassCon ( baseClass args )
                       { DerivedClass constructor body }
```

Example:

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

// Base class
class Base {
private:
    int baseValue;

public:
    // Parameterized constructor
    Base(int value) : baseValue(value) {
        cout << "Base constructor called with value: " << baseValue << endl;
    }

    // Getter method
    int getBaseValue() {
        return baseValue;
    }

    // Destructor
    ~Base() {
        cout << "Base destructor called" << endl;
    }
};

// Derived class inheriting from Base
class Derived : public Base {
private:
    int derivedValue;

public:
    // Parameterized constructor
    Derived(int baseValue, int value) : Base(baseValue), derivedValue(value) {
        cout << "Derived constructor called with value: " << derivedValue << endl;
    }

    // Getter method
    int getDerivedValue() {
        return derivedValue;
    }

    // Destructor
    ~Derived() {
        cout << "Derived destructor called" << endl;
    }
};

int main() {
    cout << "Creating a Derived object:" << endl;
    Derived derivedObj(10, 20); // Creating an object of the Derived class with
values 10 and 20

    cout << "\nBase value: " << derivedObj.getBaseValue() << endl;
    cout << "Derived value: " << derivedObj.getDerivedValue() << endl;

    cout << "\nDestroying the Derived object:" << endl;

    return 0;
}
```
Output:

```
Creating a Derived object:
Base constructor called with value: 10
Derived constructor called with value: 20

Base value: 10
Derived value: 20

Destroying the Derived object:
Derived destructor called
Base destructor called
```

--------------------------------