OntarioTech Science

Kourosh Davoudi
kourosh@uoit.ca

Week 5: Basic Topics in Classes

CSCI 1061: Programming Workshop II

Learning Outcomes

In this week, we learn:

- Public/Private Members
- Constructor/Destructors
- Scope rules and lifetime
- Class with Resources
- Friend Functions
- Static Members

Object Oriented Programming

- Object Oriented Programming Foundations:
  - **Encapsulation** ✓
  - Inheritance
  - Polymorphism

$\searrow$ Each object has

Public/Private parts

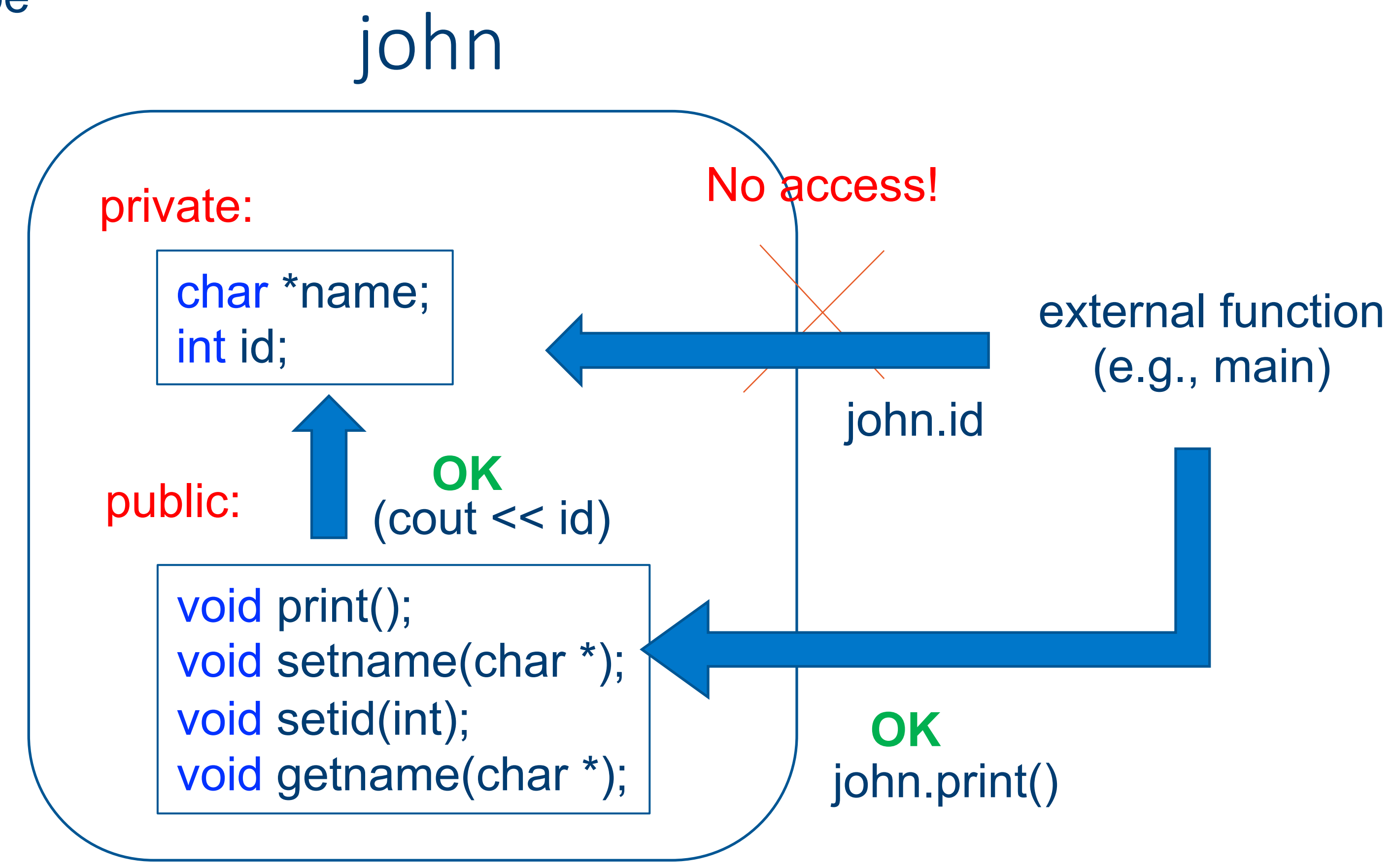
Private/Public Members

class defines a type

```
class Student{
  private:
    char *name:
    int id;
  public:
    void print();
    void setname(char *);
    void setid(int);
    void getname(char *);
};

Student john;
```

Instance of a class is an **object**;



Private/Public Members

- Private members:
  - All members in the private section can be accessed just by other member functions

- Public members:
  - All members in the public section can be accessed by any function (members or non-members)

Constructor

- Why do we need constructors?
    - Constructor provides a mechanism to take some actions automatically at the time of instantiation.

- How they are useful?

  - Object Initialization

  - Resource Allocation (e.g., Dynamic Memory Allocation)

Constructor

- When constructor is called?
  - Constructor for the object will automatically be called at the time of instantiation (no explicit call needed)

- Syntax: Constructor is a member function which :
  - Has the same name as class
  - No return value

- Why constructor for initialization?
  - Initializing an object's instance variables in a constructor ensures that the object has a well-defined state from the time of its creation.

Destructors

- Destructor provides a mechanism to take some actions at when the object lifetime is over.
  - In order to know when destructor is called, we need to know the lifetime rules.
- Syntax:
  - Name = ~ plus the name of class
  - Always no return value
  - Always no parameters

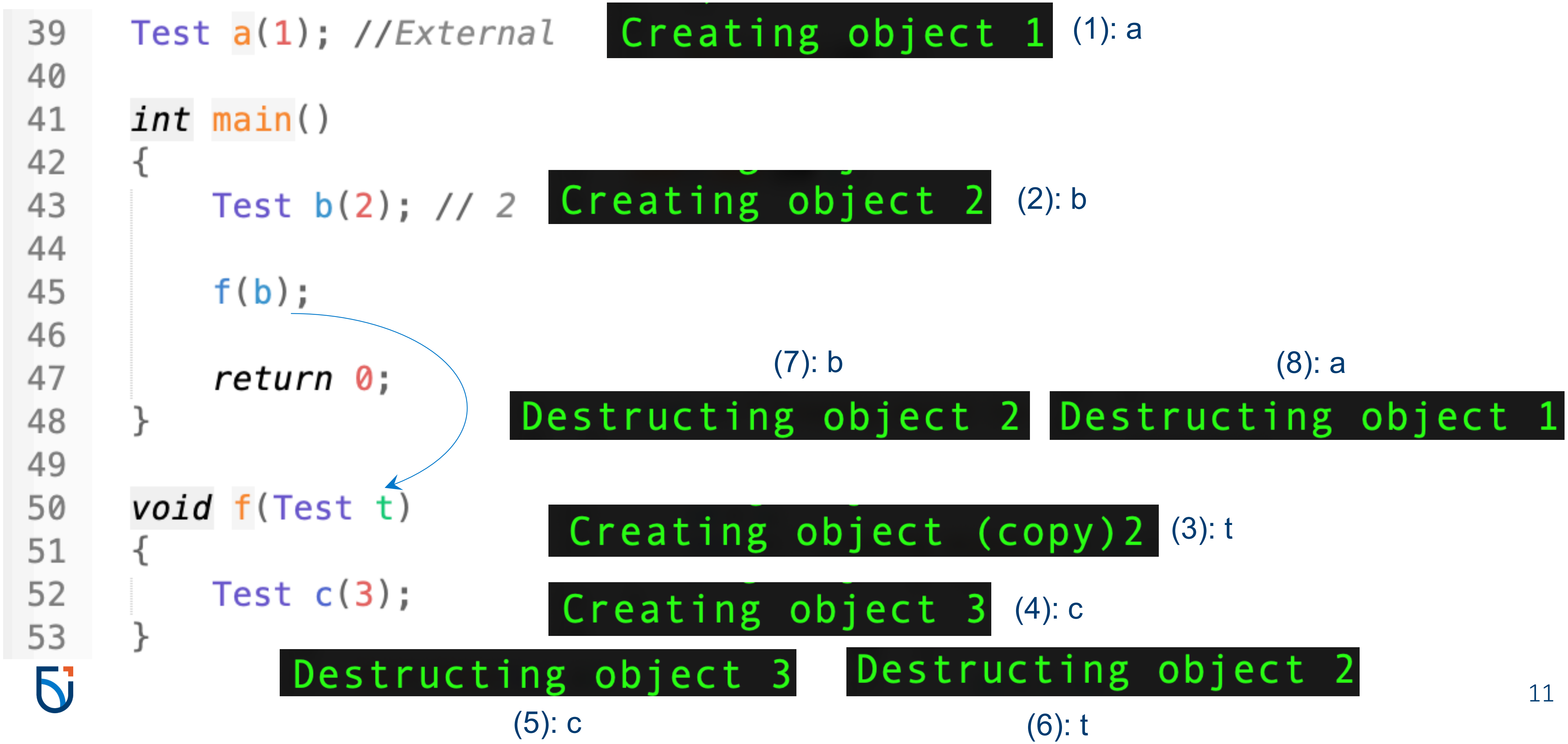
In case that several objects lifetimes are over, the order of destructor call is the reverse of creation of objects.

Scope and Lifetime Rules

- Scope:
  - Where a variable is accessible?
    - External variable: from the point that is defined till the end of file
    - Local variable: inside the block ({…}) that is defined

- Lifetime:
  - When a variable is accessible?
    - External variable: they are created at the beginning of program and exist till the end of program
    - Local variable: they are created when function is called or we enter the block ({}) and are destroyed when we return from the function or exit the block

When constructor and destructor are called?



```cpp
Test a(1); //External

int main()
{
    Test b(2); // 2

    f(b);

    return 0;
}

void f(Test t)
{
    Test c(3);
}
```

```
Creating object 1
```
(1): a

```
Creating object 2
```
(2): b

(7): b

```
Destructing object 2
```

(8): a

```
Destructing object 1
```

```
Creating object (copy)2
```
(3): t

```
Creating object 3
```
(4): c

```
Destructing object 3
```
(5): c

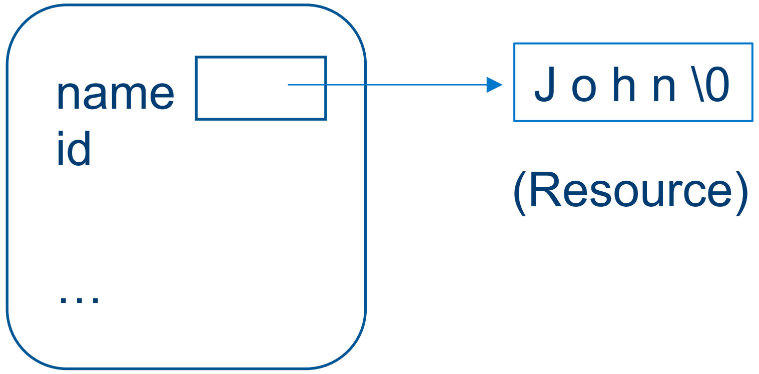
```
Destructing object 2
```
(6): t

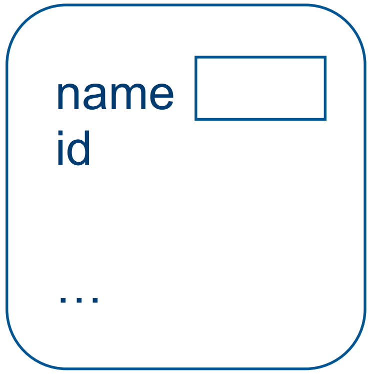
Class with Resources

- Example of resources:
  - Dynamic memory allocation
    - Allocate memory in constructor
    - Free memory in destructor

```
void Student::Student(char const *n)
{
    name = new char[strlen(n)+1];
    strcpy(name, n);
}
```

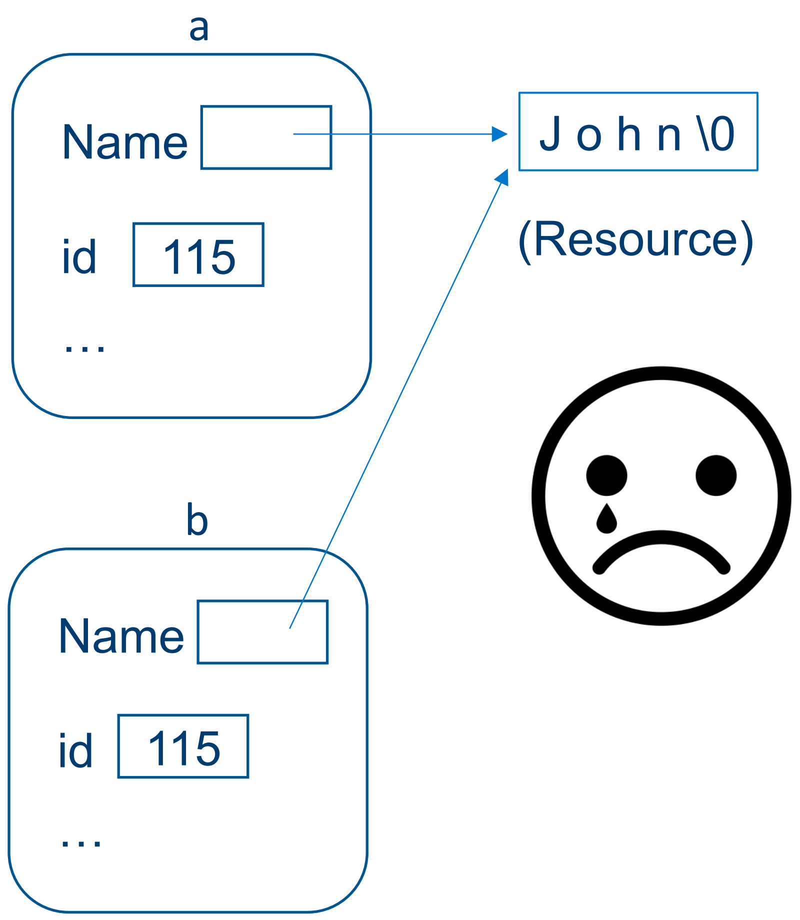
```
Student::~Student()
{
    delete [] name;
}
```





Copy Constructor for Class with Resources

- We need to do deep copy (allocating new resource and copy the information) in copy constructor.



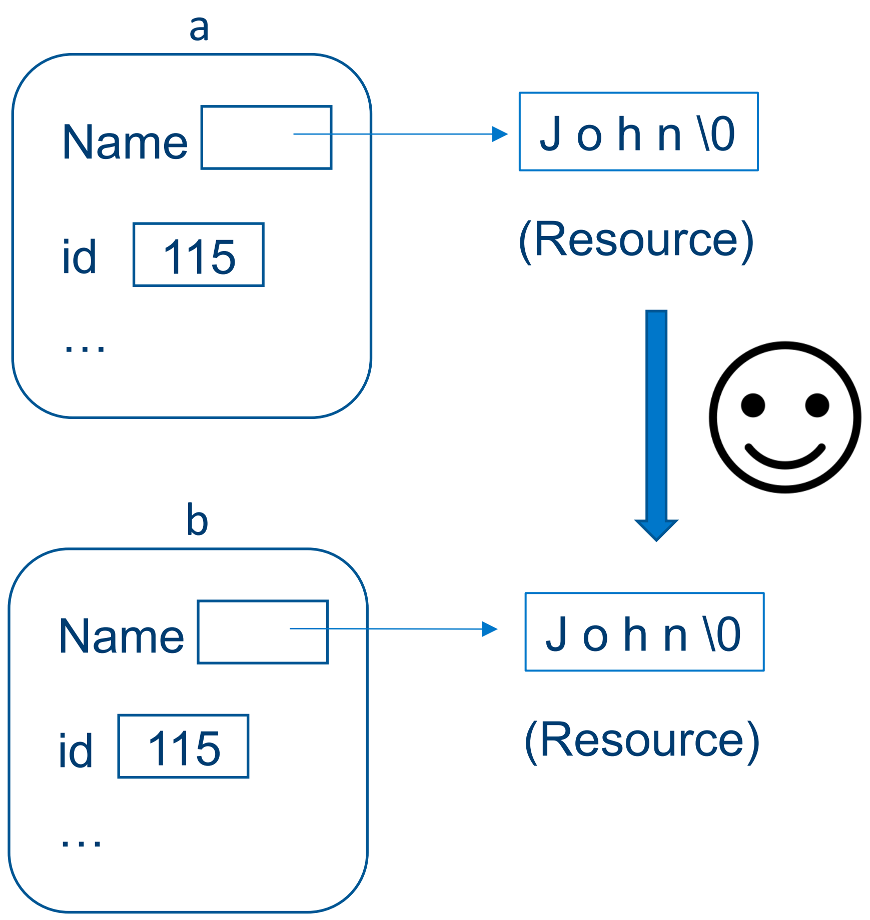
```
Student a(“John”, 115);
```

```
Student b=a;
```

default behavior is to copy element by element which is not desirable !

Copy Constructor for Class with Resources

- We need to do deep copy (allocating new resource and copy the information) in copy constructor.



```
Student a("John", 115);
```

Allocate the memory and copy the information in copy constructor

```
Student b=a;
```

Copy constructor will be called for b ☺

Friend functions

- By granting friendship status, a class lets an external helper function access to any of its private members: data members or member functions.

  Note: helper function is not the member of class

Syntax: friend + function prototype

- Is friendship is harmful for encapsulation?
  - Many believe "Yes"
  - So Why? some times efficiency is a matter

Static Members

- If you define a member as a static member, you just have one copy for all instances (objects) of the class

- You can class have access to the static object using the name of class rather then the name of object:

```
Student::help();

Student::num_student_obj;
```