



# Foundation of Computer Science: Class

Kafi Rahman

Assistant Professor

Computer Science

Truman State University



# Destructor: example

```
class Student
{
private:
    int student_ID;
    string * student_name;
public:
    Student(int id, string * name)
    { student_ID = id;
      student_name = new string;
      *student_name = *name;
    }

    ~Student()
    {
        delete student_name;
    }

    string to_string()
    { return *student_name + " (" + std::to_string(student_ID) + ");";
    }
};

int main()
{
    string name = "Harry Potter";
    Student top_student (100, &name);
    cout<< top_student.to_string() << endl;
    return 0;
}
```



# Constructors, Destructor, and Dynamically Allocated Objects

- When an object is dynamically allocated with the new operator, its constructor executes:

```
Rectangle *r = new Rectangle(10, 20);
```

- When the object is destroyed, its destructor executes:

```
delete r;
```



# JIT Quiz (1 of 4)

```
class Movie // class of a Movie
{ // private variables
    string title;
    string director;
    unsigned release_year;

public:
    Movie(string t, string d, unsigned y)
    {
        cout << "Initializing movie (" << t << ") ..." << endl;
        title = t;
        director = d;
        release_year = y;
    }
    ~Movie ()
    {
        cout << "Releasing movie (" << title << ")..." <<
endl;
    }
    // functions that can use the class variables
    void display()
    {
        cout << title << "; " << director << " (" << release_year << ")"
<< endl;
    }
}; // end of the class definition
```

```
int main()
{
    Movie normMovie("Big Fish", "Tim Burton", 2003);
    normMovie.display();

    cout << "\n\n Thanks for using the program " <<
endl;
    cout << "The program ends" << endl;
    return 0;
}
```



# JIT Quiz (2 of 4)

```
class Movie // class of a Movie
{ // private variables
    string title;
    string director;
    unsigned release_year;

public:
    Movie(string t, string d, unsigned y)
    {
        cout << "Initializing movie (" << t << ") ..." << endl;
        title = t;
        director = d;
        release_year = y;
    }
    ~Movie ()
    {
        cout << "Releasing movie (" << title << ") ..." <<
endl;
    }
    // functions that can use the class variables
    void display()
    {
        cout << title << "; " << director << " (" << release_year << ")"
<< endl;
    }
}; // end of the class definition
```

```
int main()
{
    Movie * ptrMovie = new Movie ("Harry Potter",
    "Chris Columbus", 2001);
    ptrMovie -> display();
    delete ptrMovie;

    cout << "\n\n Thanks for using the program " <<
endl;
    cout << "The program ends" << endl;
    return 0;
}
```



# JIT Quiz (3 of 4)

```
class Rectangle
{
private:
    double width;
    double length;

public:
    Rectangle(double w =0, double l=0)
    {
        width = w;
        length = l;
    }
    string to_string()
    {
        return std::to_string(width) + " "
            + std::to_string(length);
    }
};
```

```
// driver function
int main()
{
    Rectangle window;
    cout<< window.to_string() << endl;

    Rectangle main_window(10);
    cout<< main_window.to_string() << endl;

    Rectangle side_window(100, 50);
    cout<< side_window.to_string() << endl;

    return 0;
}
```



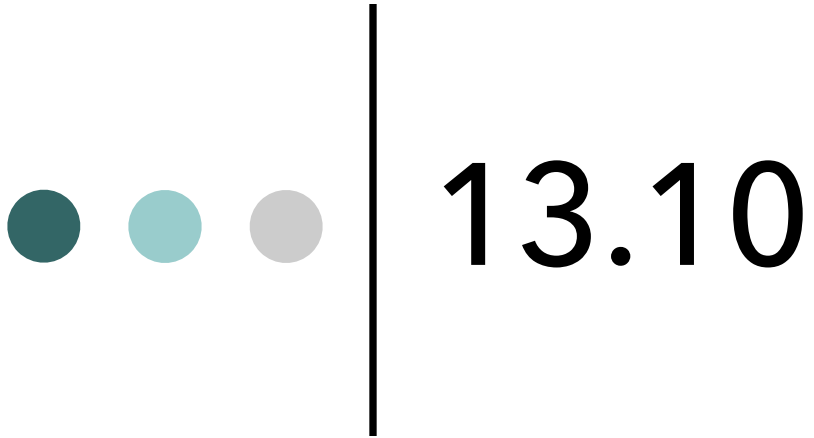
# JIT Quiz (4 of 4)

```
class Rectangle
{
private:
    double width;
    double length;

public:
    Rectangle()
    {
        width = 0;
        length = 0;
    }
    Rectangle(double w =0, double l=0)
    {
        width = w;
        length = l;
    }
    string to_string()
    {
        return std::to_string(width) + " " +
               std::to_string(length);
    }
};
```

```
// driver function
int main()
{
    Rectangle window;
    cout<< window.to_string() << endl;

    return 0;
}
```



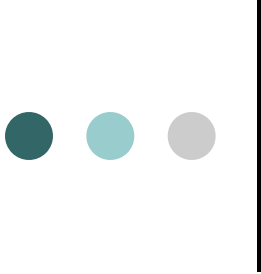
## Overloading Constructors





# Overloading Constructors

- A class can have more than one constructor
- Overloaded constructors in a class must have different parameter lists:
  - `Rectangle();`
  - `Rectangle(double);`
  - `Rectangle(double, double);`



# Only One Default Constructor and One Destructor

- Do not provide more than one default constructor for a class: one that takes no arguments and one that has default arguments for all parameters
  - `Square();`
  - `Square(int = 0);` // will not compile
- Since a destructor takes no arguments, there can only be one destructor for a class



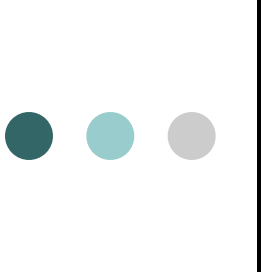
# Overloading Constructors

```
1 // This class has overloaded constructors.
2 #ifndef INVENTORYITEM_H
3 #define INVENTORYITEM_H
4 #include <string>
5 using namespace std;
6
7 class InventoryItem
8 {
9 private:
10     string description; // The item description
11     double cost;        // The item cost
12     int units;          // Number of units on hand
13 public:
14     // Constructor #1
15     InventoryItem()
16     { // Initialize description, cost, and units.
17         description = "";
18         cost = 0.0;
19         units = 0; }
20
21     // Constructor #2
22     InventoryItem(string desc)
23     { // Assign the value to description.
24         description = desc;
25
26         // Initialize cost and units.
27         cost = 0.0;
28         units = 0; }
```



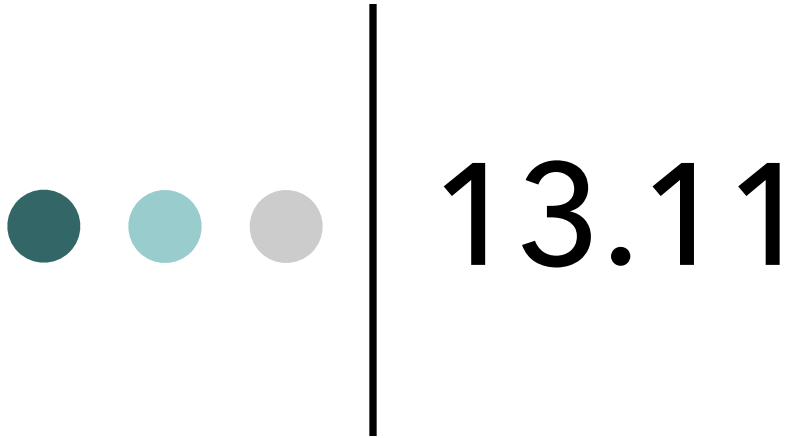
# Overloading Constructors

```
29
30     // Constructor #3
31     InventoryItem(string desc, double c, int u)
32     { // Assign values to description, cost, and units.
33         description = desc;
34         cost = c;
35         units = u; }
36
37     // Mutator functions
38     void setDescription(string d)
39     { description = d; }
40
41     void setCost(double c)
42     { cost = c; }
43
44     void setUnits(int u)
45     { units = u; }
46
47     // Accessor functions
48     string getDescription() const
49     { return description; }
50
51     double getCost() const
52     { return cost; }
53
54     int getUnits() const
55     { return units; }
56 };
57 #endif
```



# Member Function Overloading

- Non-constructor member functions can also be overloaded:
  - `void setCost(double);`
  - `void setCost(double, double);`
- Must have unique parameter lists as for constructors



Using Private Member Functions



# Using Private Member Functions

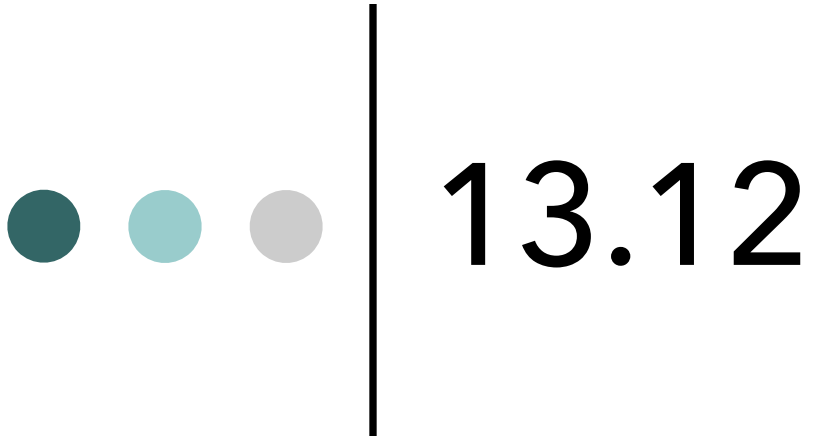
- A private member function can only be called by another member function
- It is used for internal processing by the class
  - the object of the class will not be able to call a private member function.



# Using Private Member Functions (cont)

```
20 class Circle
21 {
22     private:
23         int x, y;
24         double r;
25         // private function
26         double square(double v) |
27         {
28             return v * v;
29         }
30     public:
31         // constructor that takes three parameters
32         // also is a default constructor
33         Circle(int x_param=0, int y_param=0, double r_param=0)
34         {
35             x = x_param; y = y_param; r = r_param;
36         }
37         // calculates and returns the area of the circle
38         double getArea()
39         {
40             const double PI = 3.14159f;
41             return PI * square(r);
42         }
43     };
```





Arrays of Objects



# Arrays of Objects

- Objects can be the elements of an array:
  - `InventoryItem inventory[40];`
- Default constructor for object is used when array is defined

- If the constructor requires more than one argument, the initializer must take the form of a function call:

[illegible]



# Arrays of Objects

- It isn't necessary to call the same constructor for each object in an array:

```
InventoryItem inventory[3] = { "Hammer",  
                               InventoryItem("Wrench", 8.75, 20),  
                               "Pliers" };
```



# Accessing Objects in an Array

- Objects in an array are referenced using subscripts
- Member functions are referenced using dot notation:
  - `inventory[2].setUnits(30);`
  - `cout << inventory[2].getUnits();`

# Accessing Objects in an Array

## Program 13-13

```
1  // This program demonstrates an array of class objects.
2  #include <iostream>
3  #include <iomanip>
4  #include "InventoryItem.h"
5  using namespace std;
6
7  int main()
8  {
9      const int NUM_ITEMS = 5;
10     InventoryItem inventory[NUM_ITEMS] = {
11         InventoryItem("Hammer", 6.95, 12),
12         InventoryItem("Wrench", 8.75, 20),
13         InventoryItem("Pliers", 3.75, 10),
14         InventoryItem("Ratchet", 7.95, 14),
15         InventoryItem("Screwdriver", 2.50, 22) };
16
17     cout << setw(14) << "Inventory Item"
18         << setw(8) << "Cost" << setw(8)
19         << setw(16) << "Units On Hand\n";
20     cout << "-----\n";
```

# Accessing Objects in an Array

```
21
22     for (int i = 0; i < NUM_ITEMS; i++)
23     {
24         cout << setw(14) << inventory[i].getDescription();
25         cout << setw(8) << inventory[i].getCost();
26         cout << setw(7) << inventory[i].getUnits() << endl;
27     }
28
29     return 0;
30 }
```

## Program Output

Inventory Item	Cost	Units On Hand
Hammer	6.95	12
Wrench	8.75	20
Pliers	3.75	10
Ratchet	7.95	14
Screwdriver	2.5	22