CONSTRUCTORS AND INITIALIZER LISTS

Constructors are member functions that construct objects

Delroy A. Brinkerhoff

CONSTRUCTORS AND THEIR CHARACTERISTICS

- Constructors are member functions that build or construct objects
- They are called automatically when a new object must be constructed
  - `foo f1(123);`
  - `foo* f2 = new foo(123);`
- Two distinguishing characteristics
  - The function name is the same as the class name
  - They do not have a return type (not even void)

FIVE KINDS OF CONSTRUCTORS

Constructor	Example
Default	`class-name()`
Conversion	`class-name(type t)`
Copy	`class-name(class& o)`
Move	`class-name(class&& o)`
General	`class-name(..., ..., ...);`

INITIALIZER LIST NOTATION

- Initializer lists are only allowed with constructors
- Introduced by a colon
- Come between the argument list and the function body
- Are used to initialize member variables, often with function arguments
- Are function calls (but some behave like simple assignment)

EXAMPLE INITIALIZER LIST NOTATION

```
class fraction
{
    private:
       int    numerator;
       int    denominator;
    public:
       fraction(int n, int d)
              : numerator(n), denominator(d) {}
};
```

INITIALIZING MEMBER VARIABLES

WORKS

```
fraction::fraction(int n, int d)
{
        numerator = n;
        denominator = d;
}
```

PREFERRED

```
fraction::fraction(int n, int d)
     : numerator(n), denominator(d) {}
```

INITIALIZER LIST:
TWO-FILE ORGANIZATION

HEADER FILE

```
class fraction
{
    private:
        int     numerator;
        int     denominator;
    public:
        fraction(int n, int d);
};
```

SOURCE CODE FILE

```
fraction::fraction(int n, int d)
    : numerator(n), denominator(d)
{
        .
        .
        .
}
```

DEFAULT ARGUMENTS AND INITIALIZER LISTS

UML

- `+fraction(n: int = 0, d : int = 1)`

C++

- `fraction(int n = 0, int d = 1);`

-
  ```
  fraction(int n = 0, int d = 1)
     : numerator(n), denominator(d) {}
  ```

DEFAULT ARGUMENTS: TWO-FILE ORGANIZATION

HEADER FILE

```
class fraction
{
    private:
        int    numerator;
        int    denominator;
    public:
        fraction(int n = 0, int d = 1);
};
```

SOURCE CODE FILE

```
fraction::fraction(int n, int d)
     : numerator(n), denominator(d)
{
        .
        .
        .
}
```

LIMITS OF DIRECT INITIALIZATION

- In-class initialization does not always eliminate the need for a default constructor or default arguments
- Without a default constructor, the presence of parameterized constructors prevents creating empty fractions

```
fraction f1;
fraction* f2 = new fraction;
```

```
class fraction
{
    private:
        int    numerator = 0;
        int    denominator = 1;
    public:
        fraction() {}
        fraction(int n);
        fraction(int n, int d);
};
```