

Creating Objects

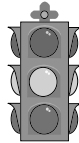
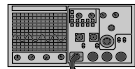
• Defining new types

- In addition to built-in (e.g., "int")
- Programmer-defined



• Limited mechanisms

- Enumerations
- Typedefs



• More powerful tool

- Classes

Object Components

• Attributes

- Characteristics, properties
- Implemented as data



• Behaviors

- Operations, messages
- Implemented as member functions



Simple Object: Date



• Attributes ("knows")

- Year (4-digit! - avoid Y2K problem)
- Month of year (1 .. 12?)
- Day of month (1 .. 31?)

• Behavior ("can do")

- Initialize its day, month, and year
- Report its day, month, or year
- Advance to next date

Date Class Definition

```

class Date
{
public:
    void SetDate(int d, int m, int y);
    int GetDay() const;
    int GetMonth() const;
    int GetYear() const;
    void Advance();
private:
    int day;
    int month;
    int year;
};

```

Annotations:

- Class name: points to `Date`
- unsigned int better?: points to `int` in `GetDay()`
- Member functions: points to the public methods
- Data members: points to the private variables
- Don't forget the semicolon!: points to the closing brace `};`

Member Function Definition

```

int Date::GetDay() const
{
    return day;
}

```

Annotations:

- Class name: points to `Date`
- Member function name: points to `GetDay`
- Function body: points to the curly braces containing `return day;`

Member Function Call

```

Date today;
...
int dy;
dy = today.GetDay();

```

Annotations:

- Object name: points to `today` in the declaration and `today` in the call
- Member function name: points to `GetDay` in the call
- Member access operator: points to `.` in the call

Types of Member Functions

• Inspector

- Returns the value of an attribute
- Also known as an “accessor”

• Mutator

- Change/set the value of an attribute

• Facilitator

- Perform some action or service
- Not everyone uses this term

Date Class Inspectors

```

class Date
{
...
  int GetDay()    const;
  int GetMonth() const;
  int GetYear()  const;
...
};

```

Return type same as data member

No arguments

“const” indicates that operation does not modify any data members

Date Class Mutator

```

class Date
{
...
  void SetDate(int d, int m, int y);
...
};

```

Usually no return value

One or more arguments

No “const”; indicates that operation may modify data members

Why a 3-arg mutator instead of three separate ones?

Date Class Facilitators

```

class Date
{
...
void Advance();
Date GetRelativeDate (int days) const;
...
};

```

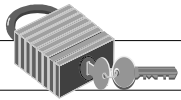
Return value and arguments as needed

"const" or not, as appropriate

Public and Private Members

- **Class members in blocks**
 - Preceded by header
 - public:
 - private:
- **Controls access to members**
 - Inside / outside class

Member Access



- **Outside class**
 - Access public data members
 - Call public member functions
- **Inside class (member functions)**
 - Access public or private data
 - Call public or private functions

Most common arrangement:

- private data members
- private/public member functions

Access Example - Outside

```

Date payday (20, 12, 1999);

payday.day = 2; // illegal

int mn = payday.month; // illegal
int dy = payday.GetDay(); // OK

```

Access Example - Inside

```

int Date::GetDay()
{
    return day; // OK
}
void Date::Advance ()
{
    day = GetDay () + 1; // OK
    ...
}

```

Inside member function, can use function name by itself if referring to same object

Initializing an Object

- **Data members**
 - Declared in class
- **How are they initialized?**
 - When creating an object of that type
- **Simple answer** (almost correct)
 - They are NOT initialized!
 - More complete answer later . . .

Date Example Revisited

```

// Define Date object
Date payday; ← Object definition
// Output day of month.
// Inspector for "day" data member.
cout << payday.GetDay() << endl;

```

What is the value of the "day" data member?

What is the output?



Seven horizontal lines for writing answers.

How to Initialize class Object?

- **Manual assignment?**
 - Using SetDate(...) function?
 - Function would set initial values
- **But what if we forget to call it?**
- **Ask compiler for help?**
 - Automatic function call
 - Invoked whenever object is created
 - "Instantiated"

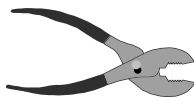


Seven horizontal lines for writing answers.

Constructor



- **Special member function**
 - Name: same as class name!
 - Return value: NONE (not even void)
- **Automatically called**
 - On each object creation
- **Initializes object state**



Seven horizontal lines for writing answers.

Date Class Constructor

```

class Date
{
public:
    Date (int d, int m, int y);
    void SetDate (int d, int m, int y);
    int GetDay () const;
    int GetMonth () const;
...
private:
    int day;
    int month;
    int year;
};

```

Constructor

Initializes data members

Constructor Definition

```

Date:Date (int d, int m, int y)
{
    day = d;
    month = m;
    year = y;
}

```

Class name

Constructor name

Data member initialization

Date Object Definition

```

Date payday (20, 12, 1999);
Date new_years (1, 1, 1997);

```

Class name

Object name

Initial values

What if we don't have initial values to specify?
"Date birthday;"

Equivalent:

```

int x = 3;
int y (5);

```

Default Initialization Options

```

class Date
{
public:
    Date (int d=1, int m=1, int y=2000);
    ...
};

```

← Default argument values

```

class Date
{
public:
    Date (); ← "No arg" constructor
    ...
};

```

Constant Data Objects

Constant type Initial values

```

const Date fourth (4, 7, 1776);

```

- Object cannot be modified
 - After it has been initialized
- But constructor still executes
 - Part of initialization

OO Terminology and C++

- **Object** - instance of a class
- **Method** - member function
- **Attribute** - data member
- **Message** - member function
- **Interface** - public members