Introduction to Classes and Objects

How to manage data and actions together.
Slides #10: Chapter 7.1-7.5

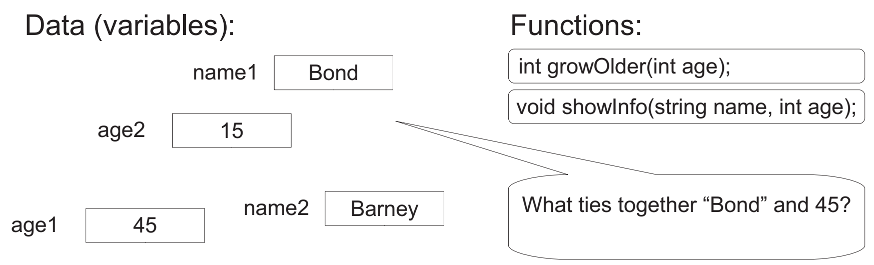
Topics

1) What is an object? What is a class?
2) How can we use objects?
3) How do we implement the functions of a class?

Objects

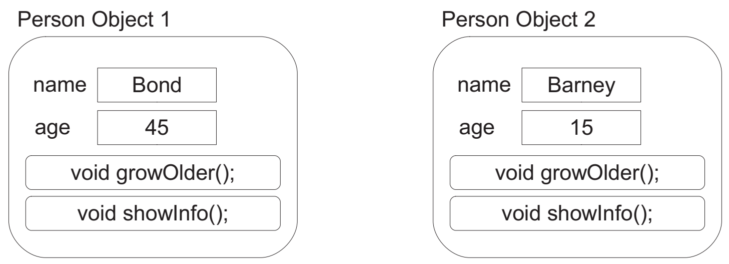
Procedural Programming

- Procedural Programming
  -
  -



Object Oriented Programming

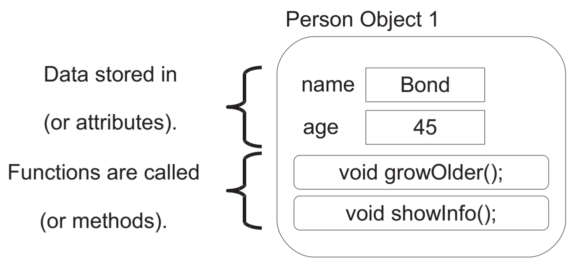
- Object Oriented Programming
  -
  -
    - The objects ties together its data.



Object

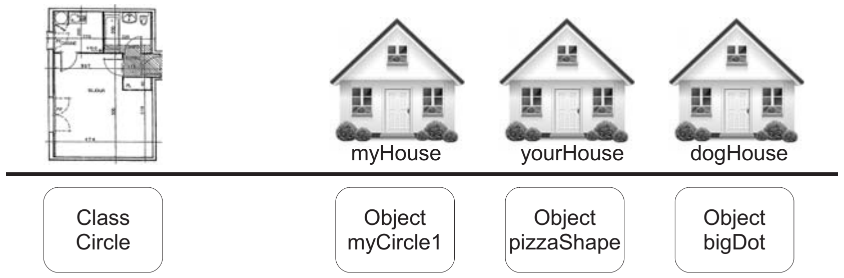
- Object:

  data and functions within a single unit.



Classes and Objects

-
  - Think of it as the blue print for a house.
    The blue print lays out the details for a type of house.
-
  - like houses which have been built from the blue print.



Review

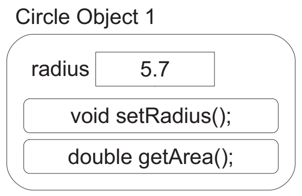
- How does object oriented programming help organize the program's data?
- Write a very short sentence which express the relationship between objects and classes.

Implementing a class.

Circle Class

Model a circle using an object

-
  - Store the circle's radius.
-
  - Set its radius.
  - Calculate its area.



Class Name `Circle`
Member variables `double radius;`

Member functions
`void setRadius(double r) {`
`   radius = r;`
`}`
`double getArea() {`
`   return 3.15 * pow(radius, 2);`
`}`

Circle class

```
class Circle {
private:
      double radius;
public:
      void setRadius(double r) {
            radius = r;
      }
      double getArea() {
            return 3.15 * pow(radius, 2);
      }
};
```

General Form:
class <Name> { ... };

Class definition inside {...}

Note that...

Very common error!

Circle class

```
class Circle {
private:
      double radius;
public:
      void setRadius(double r) {
            radius = r;
      }
      double getArea() {
            return 3.15 * pow(radius, 2);
      }
};
```

```
private:
      ...
public:
      ...
```

control if the items listed below it are:
private:
public: usable inside and outside the class.

Note colon (':') after private or public.

Circle class

```
class Circle {
private:
    double radius;
public:
    void setRadius(double r) {
        radius = r;
    }
    double getArea() {
        return 3.15 * pow(radius, 2);
    }
};
```

Member Variables:

Usually they are private.

Can have any number of member variables.

Circle class

```
class Circle {
private:
    double radius;
public:
    void setRadius(double r) {
        radius = r;
    }
    double getArea() {
        return 3.15 * pow(radius, 2);
    }
};
```

Member Functions:
usually public so they can be called from both inside and outside the class.

Member functions can access...

Able to declare any number of member functions.

Using the Circle class

```
Size of small: 113.4
Size of med:   154.34
```

```
class Circle {
private:
    double radius;
public:
    void setRadius(double r) {
        radius = r;
    }
    double getArea() {
        return 3.15 * pow(radius, 2);
    }
};
```

```
int main() {
    // Create the 2 pizza objects (Circles)
    Circle pizzaSmall,
           pizzaMed;

    // Setup the size
    pizzaSmall.setRadius(6.0);     // 12"
    pizzaMed.setRadius(7.0);       // 14"

    // Output the area
    cout << "Size of small: "
        << pizzaSmall.getArea() << endl;
    cout << "Size of med:   "
        << pizzaMed.getArea() << endl;

    return 0;
}
```

Note we only call the

we never access the private attributes.

Member access

```
class Circle {
private:
    double radius;
public:
    void setRadius(double r) {
        radius = r;
    }
    double getArea() {
        return 3.15 * pow(radius, 2);
    }
};
```

```
int main() {
    // Create the 2 pizza objects (Circles)
    Circle pizzaSmall,
           pizzaMed;

    // Setup the size
    pizzaSmall.setRadius(6.0);     // 12"
    pizzaMed.setRadius(7.0);       // 14"

    ...
}
```

Inside the class, we can access

public or private,
no dot-operator ('.') required.

Outside the class, access public member **functions** using:
object.memberFunct().

or public member **variables** using:
object.memberVar

Review

- Complete this code by creating Circle object named cropCircle1 of radius 100, and <u>output</u> its area.

```
int main () {




    return 0;
}
```

Encapsulation

Encapsulation

- Interface:
  –
- Encapsulation:
  –
  - External code must use the class' interface.
  - Benefit: Don't have to understand internals of the class in order to use it.
    - Ex: cin/cout
- Example:
  - With the Circle class, you cannot directly change the value of radius; you must use setRadius().

Encapsulation: limited access

- An object's attributes are most often...
  - How can we access a private member variable?

    Ex: Read a circle's radius?
- From outside the class we <u>cannot</u> do:

```
Circle myCircle;
myCircle.setRadius(42);
cout << myCircle.radius;
```

(and is <u>impossible</u> if radius is a private member).

Accessors and Mutators

- Accessors
  - Usually of the form getX(), where X is the attribute.
  - Also called getters.
  - Ex: getRadius(), getHeight(), getColour().
- Mutators
  - usually of the form setX(), where X is the attribute.
  - Also called setters.
  - Ex: setRadius(), setHeight(), setColour()
  - Have mutators verify new value is valid!

Demo & Review

- Assume you are given a complete Die class implementation. Use it to create a 6 sided die; roll it and <u>output</u> the value.

Class name: Die

Private Member Variables:
int numSides;
int faceValue;

Public Member Functions
void setNumSides (int value);
int getNumSides();
int roll();
int getFaceValue();

UML Class Diagram

-
  - UML: Unified Modelling Language
- Draw the class as a rectangle containing three parts:
  - Class name
  - Attribute : type
  - Method(parameters) : return-type
- + means public, - means private

Die
- numSides : int - faceValue : int
+ setNumSides (value : int) : void + getNumSides() : int + roll() : int + getFaceValue() : int

Member Functions

Where to define member functions

- Member functions can be defined in two places:

Inside the class' {...}

```
class Circle {
private:
      double radius;
public:
      void setRadius(double r) {
            radius = r;
      }
      double getArea() {
            return 3.15 * pow(radius, 2);
      }
};
```

After the class' {...}
Non-inline (normal).

```
class Circle {
private:
      double radius;
public:
      void setRadius(double r);
      double getArea();
};

void Circle::setRadius(double r) {
      radius = r;
}
double Circle::getArea() {
      return 3.15 * pow(radius, 2);
}
```

:: is the

Method comments

- Must comment each class describing what it does.
- Must comment each public member function:
  -
  - (if any)
  - (if not void)

```
/************************************
* Set the circle's radius to r.
*       r: new radius; should be >=0.
************************************/
void Circle::setRadius(double r) {
   radius = r;
}
/************************************
* Calculate the circle's area.
* return: the area.
************************************/
double Circle::getArea() {
   return 3.15 * pow(radius, 2);
}
```

Review

- Write a getRadius() member function for the Circle class. Make it non-inline, and add a comment block.

Summary

- Object Oriented Programming:
  - Classes are the blue prints.
  - Objects are the instances.
- Classes have access specifiers: public and private.
  - Encapsulation prevents access to private attributes/methods from outside the class.
- Functions normally defined outside the class.
  - Ex: int Circle::getRadius() { return radius;}
- Inline functions defined inside the class.
- Comment all member functions.