

Classes and Objects: Members, Visibility

Computer Science and Engineering • College of Engineering • The Ohio State University

Lecture 4

Object-Oriented Programming

Computer Science and Engineering • The Ohio State University

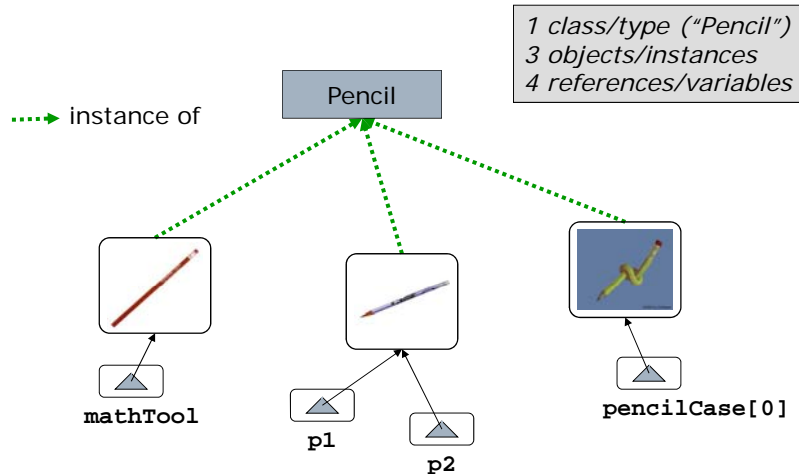
- Fundamental component is an *object*
 - A running program is a collection of objects
- An object encapsulates:
 - State (ie data)
 - Behavior (ie how state changes)
- Each object is an instance of a *class*
 - Class declaration is a blueprint for objects
 - A class is a component type
 - eg Stack, String, Partial_Map, Sorting_Machine
 - An object is an instance of that component
 - Resolve:

```
object Pencil mathTool;
```
 - Java:

```
Pencil mathTool = new Pencil();
```

Graphical View of Instances

Computer Science and Engineering • The Ohio State University



Good Practice: Files and Classes

Computer Science and Engineering • The Ohio State University

- Declare one class per file
- Give file the same name as the class declaration it contains
 - class HelloWorldApp declaration appears in HelloWorldApp.java
 - class Pencil is defined in Pencil.java

Example Class Declaration

Computer Science and Engineering ■ The Ohio State University

```
class Pencil {
    boolean hasEraser;
    String color;
    int length;

    int sharpen (int amount) {
        length = length - amount;
        return length;
    }

    String getDescription () {
        if (length < 15) {
            return "small: " + color;
        }
        else {
            return "large: " + color;
        }
    }
}
```

Members

Computer Science and Engineering ■ The Ohio State University

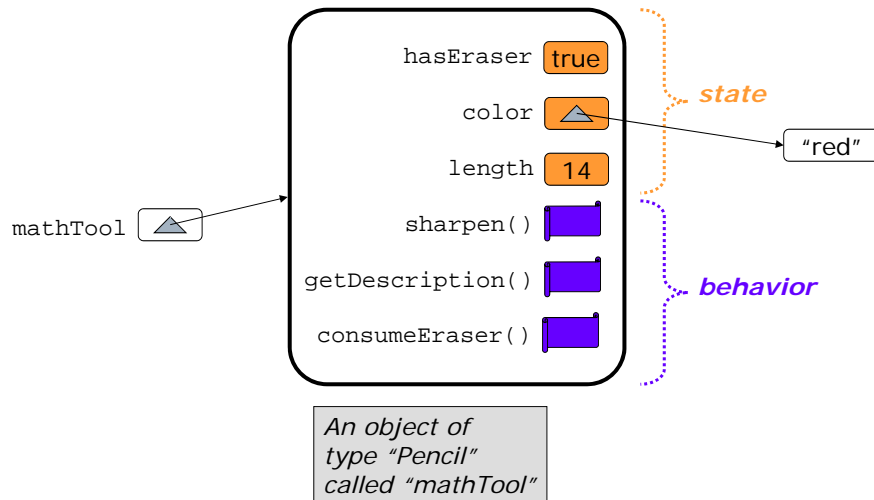
- Two kinds of members in a class declaration
 - Fields, ie data (determine the *state*)

```
boolean hasEraser;
String color;
int length;
```
 - Methods, ie procedures (*access/modify* the state)

```
int sharpen (int amount) {
    length = length - amount;
    return length;
}
```
- (Much later: nested classes and nested interfaces)

Graphical View of Object

Computer Science and Engineering ■ The Ohio State University



Object Creation and Deletion

Computer Science and Engineering ■ The Ohio State University

- Explicit object creation with **new**();

```
java.util.Date d = new java.util.Date();
Integer count = new Integer(34);
Pencil p1 = new Pencil("red");
```
- Unlike C/C++, memory is *not* explicitly freed
 - References just go out of scope (become null)

```
{
    //create a Date object (called d)
    java.util.Date d = new java.util.Date();
    . . .
} //d out of scope, object is unreachable
```
 - *Automatic* garbage collection (eventually) deletes unreachable objects

Initialization of an Object's Fields

Computer Science and Engineering • The Ohio State University

- Implicit: Default initial values based on type
 - eg boolean is false, reference type is null
 - `boolean hasEraser; //implicitly false`
- Explicit: Initialization with field declaration
 - `int length = 14;`
- Special method: "constructor"
 - Syntax: name is same as class, no return type
 - ```
class Pencil {
 String color;
 Pencil (String c) {
 color = c;
 }
}
```
  - Invoked by `new()`, so can have parameters
  - Runs *after* implicit/explicit field initialization

## Default Initial Values

Computer Science and Engineering • The Ohio State University

- For fields only
- Does not apply to local variables

| Type      | Default  |
|-----------|----------|
| boolean   | false    |
| byte      | 0        |
| short     | 0        |
| int       | 0        |
| long      | 0L       |
| float     | 0.0f     |
| double    | 0.0d     |
| char      | '\u0000' |
| reference | null     |

## Example Constructor

Computer Science and Engineering ■ The Ohio State University

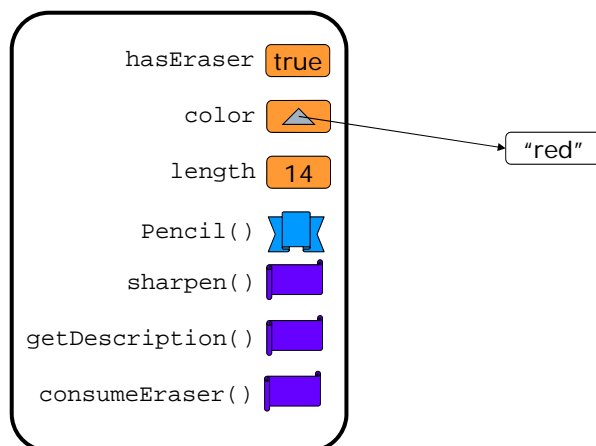
```
class Pencil {
 boolean hasEraser;
 String color;
 int length = 14;

 Pencil (String c) {
 color = c;
 hasEraser = (length >= 10);
 }

 . . . same methods as before . . .
}
```

## Graphical View of Object

Computer Science and Engineering ■ The Ohio State University



## Good Practice: Establish Invariant

Computer Science and Engineering • The Ohio State University

- Ensures clause of a constructor:  
establishes the convention  
(representation invariant) for this  
instance
  - What is true of the state for all instances?
  - eg All long pencils have erasers  
`length >= 10 ==> hasEraser`
  - So the state (false, "green", 14) is not valid
- A constructor can call other methods of  
its own object
  - Danger! Convention (representation  
invariant) might not hold at call point

## Visibility

Computer Science and Engineering • The Ohio State University

- Members can be private or public
  - member-by-member declaration  
`private String color;`  
`public int length;`  
`public int sharpen (int amount) { . . . }`
- Private members
  - Can be accessed only by instances of same class
  - Provide concrete implementation / representation
- Public members
  - Can be accessed by any object
  - Provide abstract view (client-side)

# Example

```
class Pencil {
 private String color;
 private int length = 14;
 private boolean isValid(String c) {...}
 public Pencil(String c, int l) {...}
 public String toString() {...}
 public void setColor(String c) {...}
}
```

```
class CreatePencil {
 public void m() {
 Pencil p = new Pencil("red", 12);
 p.setColor("blue");
 p.color = "blue";
 }
}
```

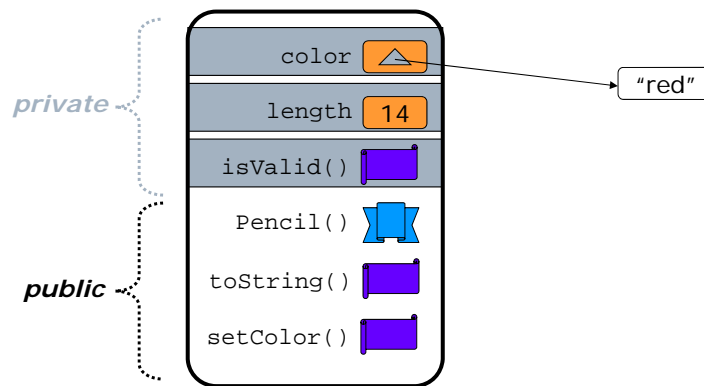
OK



Compile-time Error



# Graphical View of Member Visibility



## Example

Computer Science and Engineering ■ The Ohio State University

- See PencilA.java
  - Concrete state (ie representation) is hidden from clients
  - Abstract state (ie client-side view) is accessed and manipulated through public methods
- See PencilB.java
  - Different representation
  - Exact same behavior as far as the outside world is concerned

## Good Practice: Member Declarations

Computer Science and Engineering ■ The Ohio State University

- Group member declarations by visibility
  - Java's convention: private members at top
- No fields should be public
  - Common (bad) idiom: Public "accessor" methods for getting and setting private fields (aka getters/setters)

```
class Pencil {
 private int length;
 public int getLength() { . . . }
 public void setLength(int) { . . . }
}
```
  - Better idiom: Provide public members for observing and controlling *abstract state*
    - Recall from Resolve: "Client view first"
  - Eg PencilA and PencilB should have *exactly the same accessors* (including signatures)

## Using Fields & Invoking Methods

Computer Science and Engineering • The Ohio State University

- Syntax: *objectreference.member*

```
p.color = "red";
p.toString().length();
```

- Reference is implicit inside same object

```
class Pencil {
 private String color;
 public Pencil() {
 color = "red";
 }
}
```

- Explicit reference to same object available as *this* keyword (from within the object itself)

```
this.color = "red";
```

## Good Practice: Formal Arguments

Computer Science and Engineering • The Ohio State University

- Constructor arguments that are used directly to set object fields can be given the same name as the field
  - Formal argument "hides" class field variable
  - Refer to class field variable using explicit *this*

```
class Pencil {
 private int length;
 Pencil(int length) {
 this.length = length;
 }
}
```

# Method Overloading

- A class can have more than one method with the same name as long as they have different *parameter lists*

```
class Pencil {
 . . .
 public void setPrice(float newPrice) {
 price = newPrice;
 }
 public void setPrice(Pencil p) {
 price = p.getPrice();
 }
}
```

- How does the compiler know which method is being invoked?
  - Answer: it compares the number and type of the parameters and uses the matched one  
`p.setPrice(3.4);`
- Differing *only* in return type is not allowed

# Multiple Constructors

- Default constructor: no arguments
  - Fields initialized explicitly in declaration or implicitly to language-defined initial values
  - Provided automatically *only* if no constructor defined explicitly

```
class Pencil {
 String color; //initialized implicitly to null
 int length = 14; //initialized explicitly
 ...
}
```

- Another constructor: one same-class argument  
`Pencil (Pencil p) { . . . }`
- One constructor can call another with *this()*
  - If another constructor called, must be the first statement  
`Pencil (Pencil p) {  
 this(p.color); //must be 1st line  
 length = 10;  
}`

# Summary

Computer Science and Engineering • The Ohio State University

- Classes and objects
  - Class declarations and instantiations
- Instance members
  - Fields, ie state
  - Methods, ie behaviors
- Constructors
- Visibility
  - private: Visible only to instances of same class
  - public: Visible to instances of any class
- Overloading
  - Multiple implementations of same method name
  - Distinguished by formal parameter types