CSE 1325

Week of 10/03/2022

Instructor : Donna French

```
package accountdemo;

public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Instantiating on Object

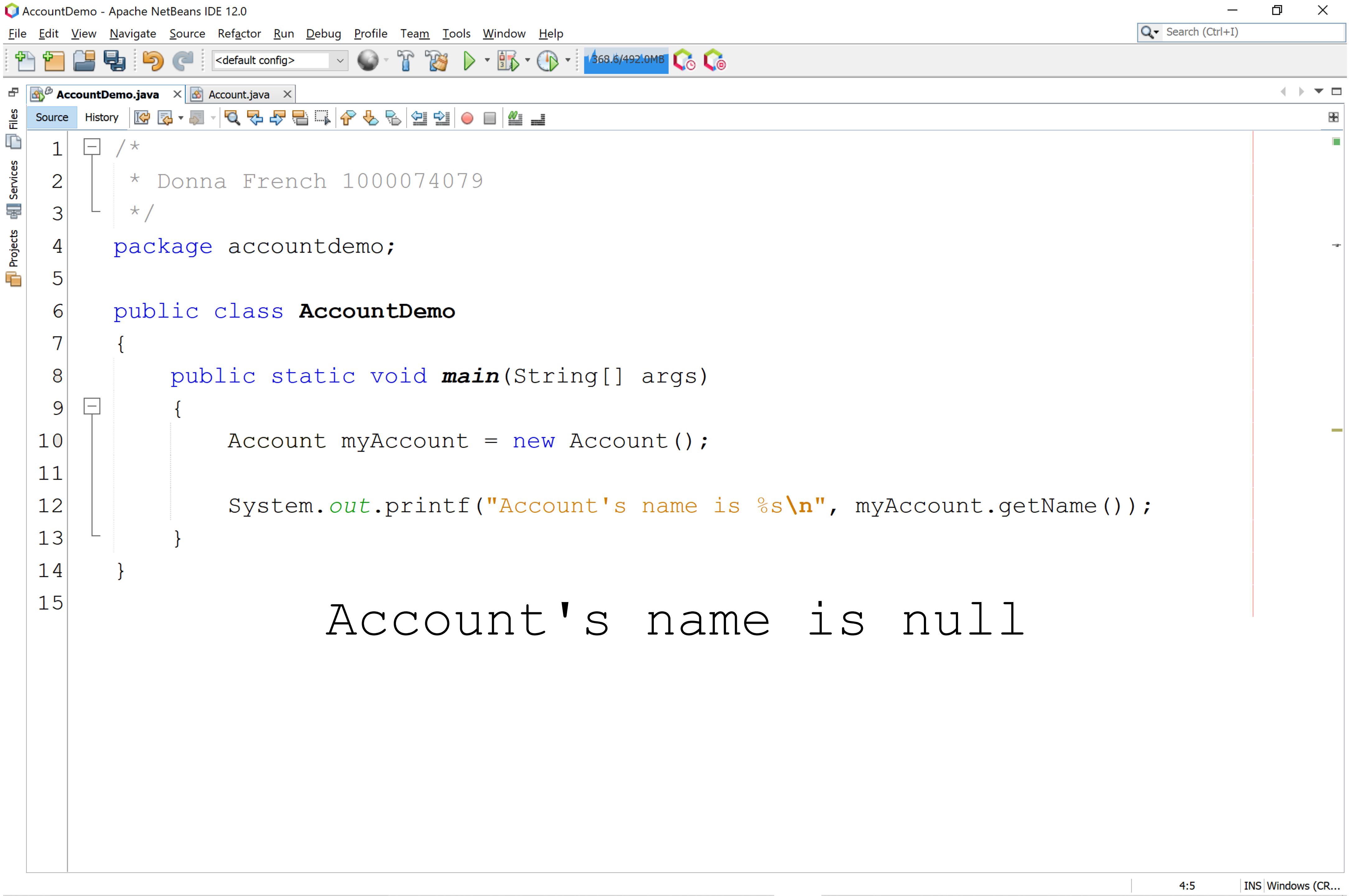
Let's make a program that can use our new object.

We need to instantiate the new object

and

invoke the instance members.

We'll do that in `AccountDemo.java`



```
/*
 * Donna French 1000074079
 */
package accountdemo;

public class AccountDemo
{
    public static void main(String[] args)
    {
        Account myAccount = new Account();

        System.out.printf("Account's name is %s\n", myAccount.getName());
    }
}

```

```
Account's name is null
```

```java
Scanner in = new Scanner(System.in);

Account myAccount = new Account();

System.out.printf("Initial name is %s\n", myAccount.getName());

System.out.print("Please enter account name ");
String theName = in.nextLine();

myAccount.setName(theName);

System.out.printf("Account's name is %s\n", myAccount.getName());
```

```
Initial name is null
Please enter account name My Piggy Bank
Account's name is My Piggy Bank
```

```
Scanner in = new Scanner(System.in);

Account myAccount = new Account();

System.out.printf("Initial name is %s\n", myAccount.getName());

System.out.print("Please enter account name ");
String theName = in.nextLine();

myAccount.setName(theName);

System.out.printf("Account's name is %s\n", myAccount.getName());
```

Instantiates a new object named `myAccount` of type `Account`

```
Initial name is null
Please enter account name My Piggy Bank
Account's name is My Piggy Bank
```

A constructor is similar to a method but is called implicitly by the `new` operator to initialize an object’s instance variables at the time the object is created.

```
Scanner in = new Scanner(System.in);

Account myAccount = new Account();

System.out.printf("Initial name is %s\n", myAccount.getName());

System.out.print("Please enter account name ");
String theName = in.nextLine();

myAccount.setName(theName);

System.out.printf("Account's name is %s\n", myAccount.getName());
```

```
Initial name is null
Please enter account name My Piggy Bank
Account's name is My Piggy Bank
```

A method call supplies values—known as arguments—for each of the method’s parameters.

Each argument’s value is assigned to the corresponding parameter in the method header.

```
Scanner in = new Scanner(System.in);

Account myAccount = new Account();

System.out.printf("Initial name is %s

System.out.print("Please enter accoun
String theName = in.nextLine();

myAccount.setName(theName);

System.out.printf("Account's name is
```

```
Initial name is null
Please enter account name My Piggy Ba
Account's name is My Piggy Bank
```

Why did the initial name print as "`null`"?

Local variables are not automatically initialized.

Every instance variable has a default initial value—a value provided by Java when you do not specify the instance variable’s initial value.

The default value for an instance variable of type `String` is `null`.

```
Scanner in = new Scanner(System.in);

Account myAccount = new Account();

System.out.printf("Initial name is %s\n", myAccount.getName());

System.out.print("Please enter account name ");
String theName = in.nextLine();

myAccount.setName(theName);

System.out.printf("Account's name is %s\n", myAccount.getName());
```

```
Initial name is null
Please enter account name My Piggy Bank
Account's name is My Piggy Bank
```

To call a method of an object, follow the object name with a dot separator, the method name and a set of parentheses containing the method's arguments.

OOP Vocabulary

UML

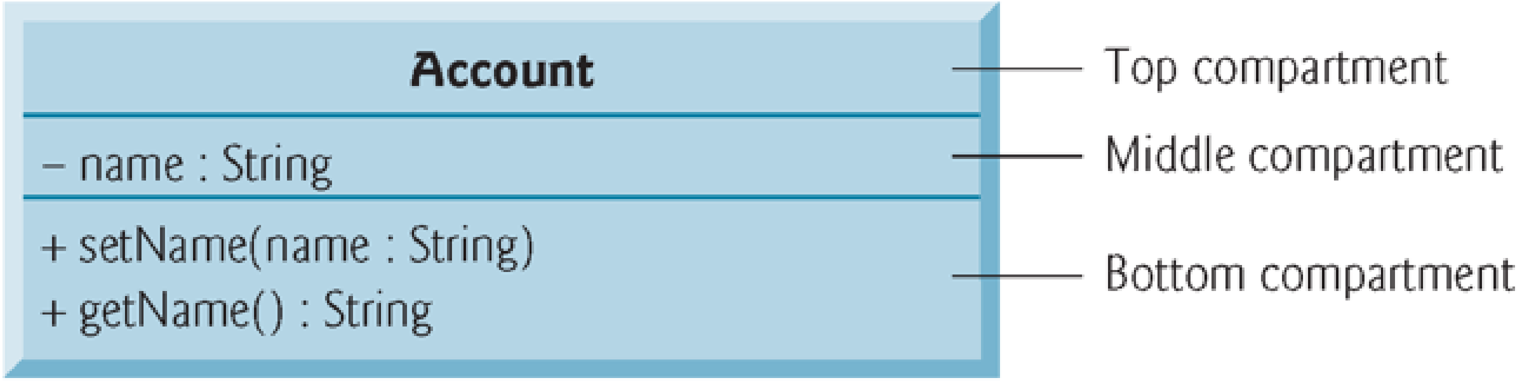
Unified Modeling Language

widely used graphical scheme for modeling object-oriented systems

We will focus only on class diagrams

UML

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```



Account

– name : String

+ setName(name : String)
+ getName() : String

Top compartment

Middle compartment

Bottom compartment

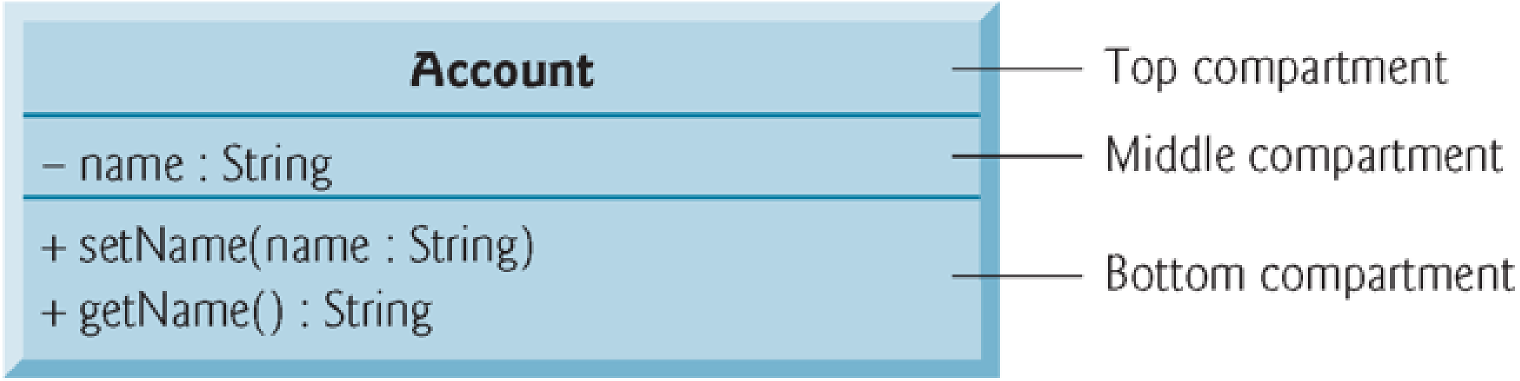
UML

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Top Compartment

In the UML, each class is modeled in a class diagram as a rectangle with three compartments.

The top one contains the class's name centered horizontally in boldface.



Account

– name : String

+ setName(name : String)
+ getName() : String

Top compartment

Middle compartment

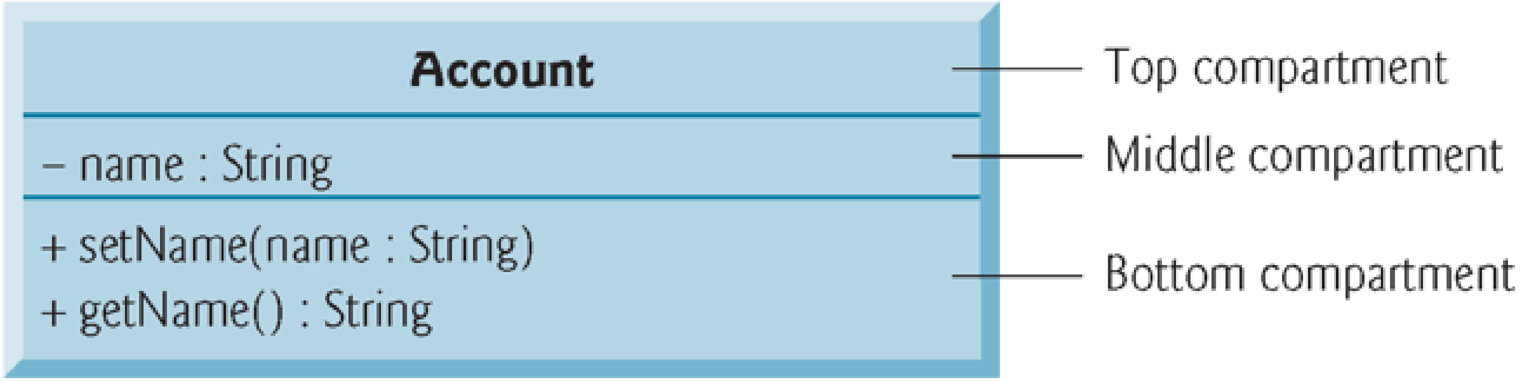
Bottom compartment

UML

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Middle Compartment

The middle compartment contains the class’s attributes, which correspond to instance variables in Java.

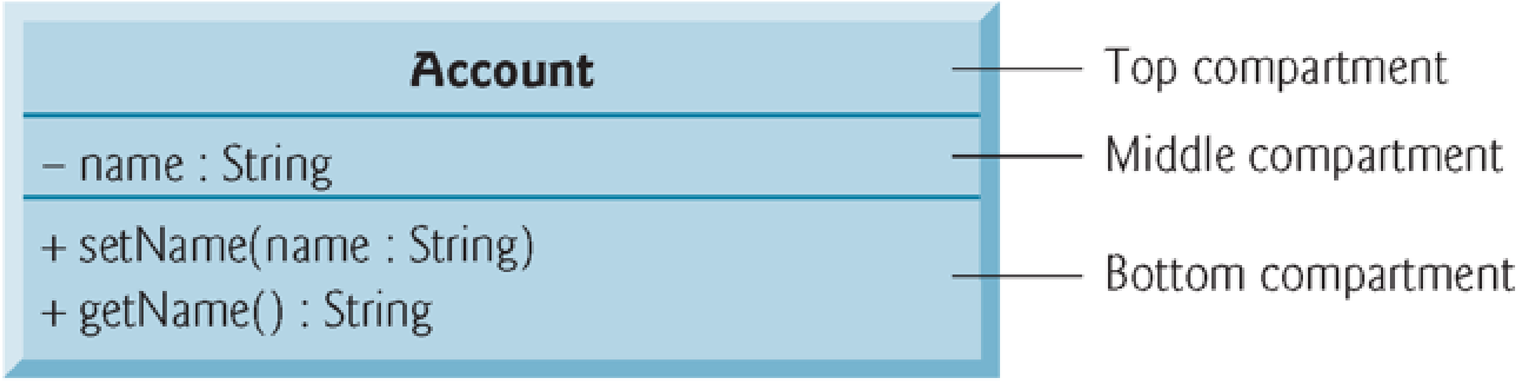


UML

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Bottom Compartment

The bottom compartment contains the class's operations, which correspond to methods and constructors in Java.



Bottom Compartment

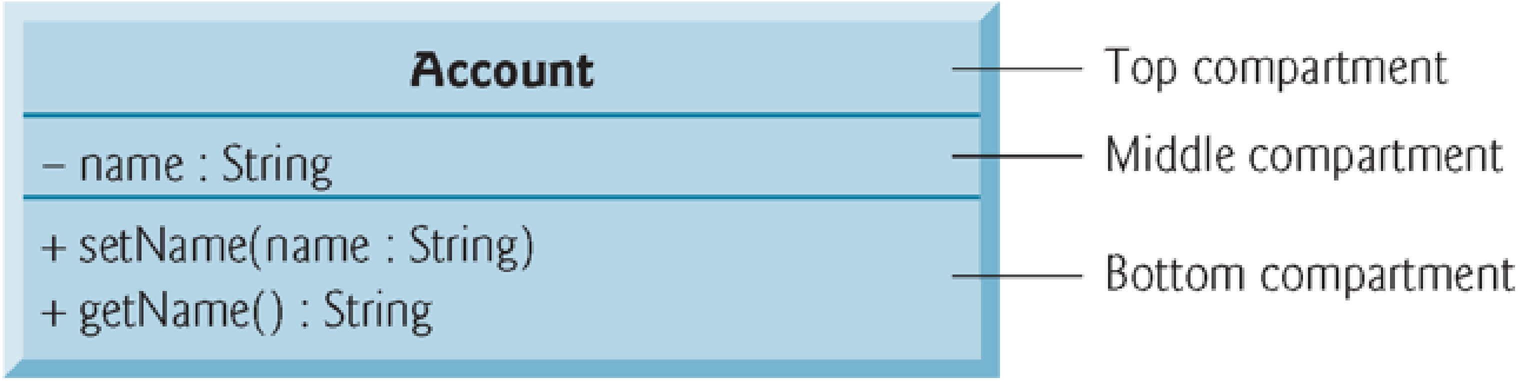
UML

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

The UML represents instance variables as an attribute name, followed by a colon and the type.

Private attributes are preceded by a minus sign (–) in the UML.

Public attributes are preceded by a plus sign (+) in the UML.



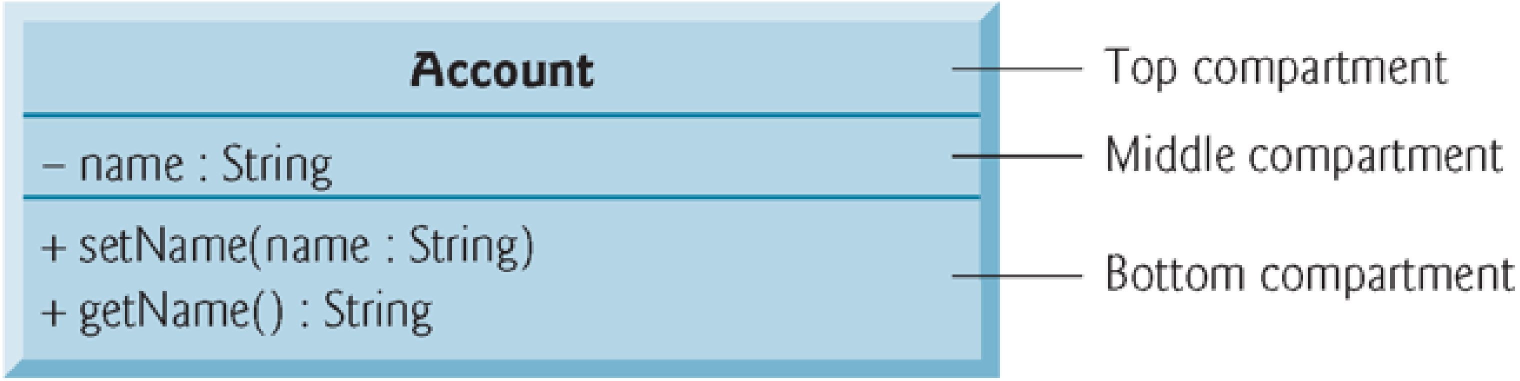
UML

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Bottom Compartment

The UML models operations by listing the operation name followed by a set of parentheses.

A plus sign (+) in front of the operation name indicates that the operation is a public one in the UML (i.e., a public method in Java).



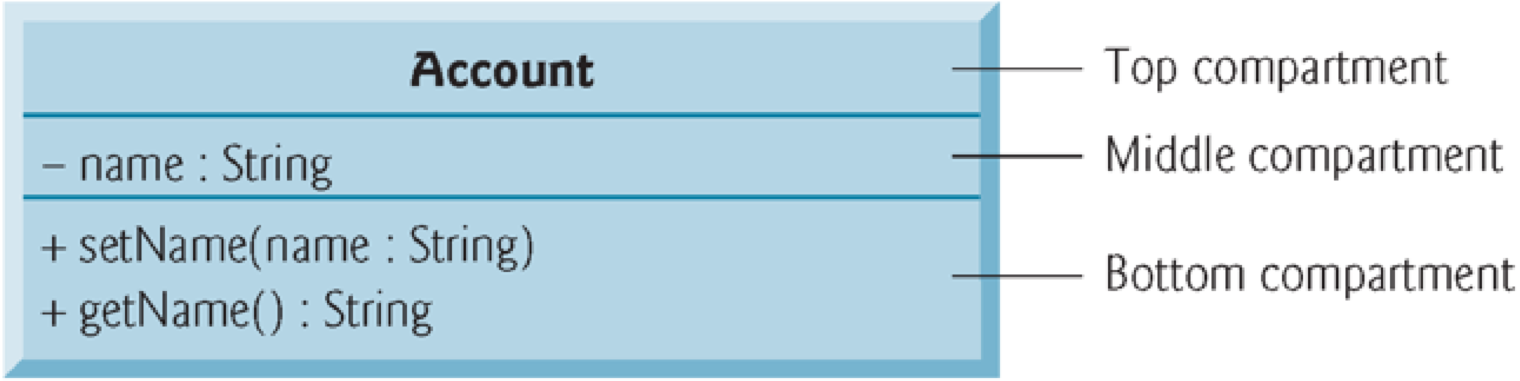
UML

Return Types

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

UML indicates an operation’s return type by placing a colon and the return type after the parentheses following the operation name.

UML class diagrams do not specify return types for operations that do not return values.

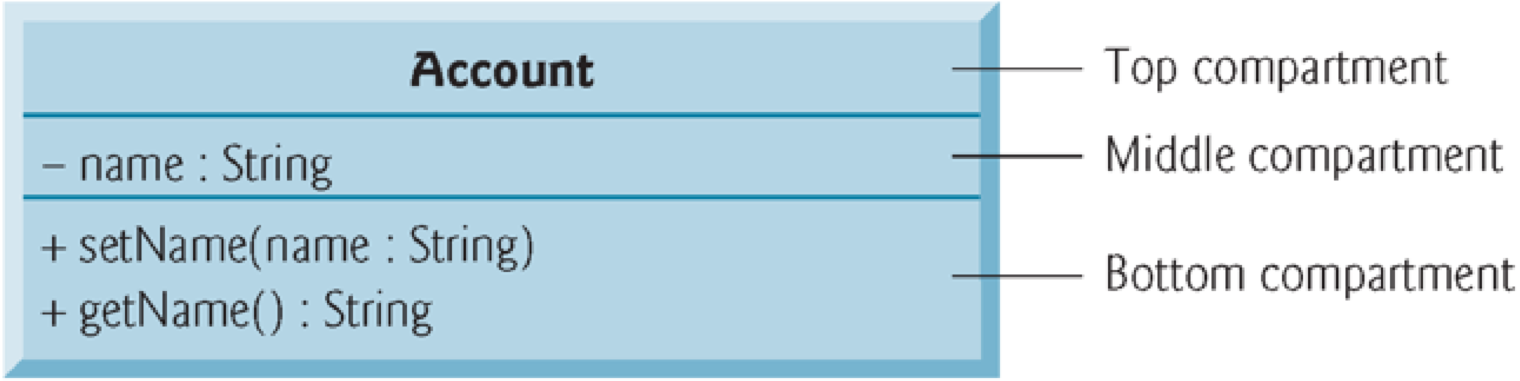


UML

Parameters

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

UML models a parameter of an operation by listing the parameter name, followed by a colon and the parameter type between the parentheses after the operation name



Account

– name : String

+ setName(name : String)
+ getName() : String

Top compartment

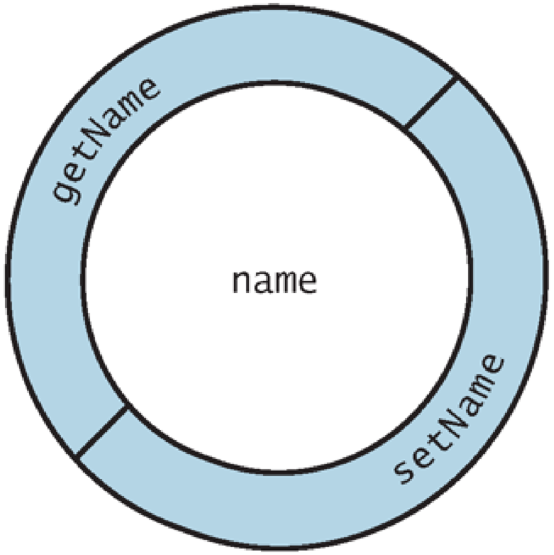
Middle compartment

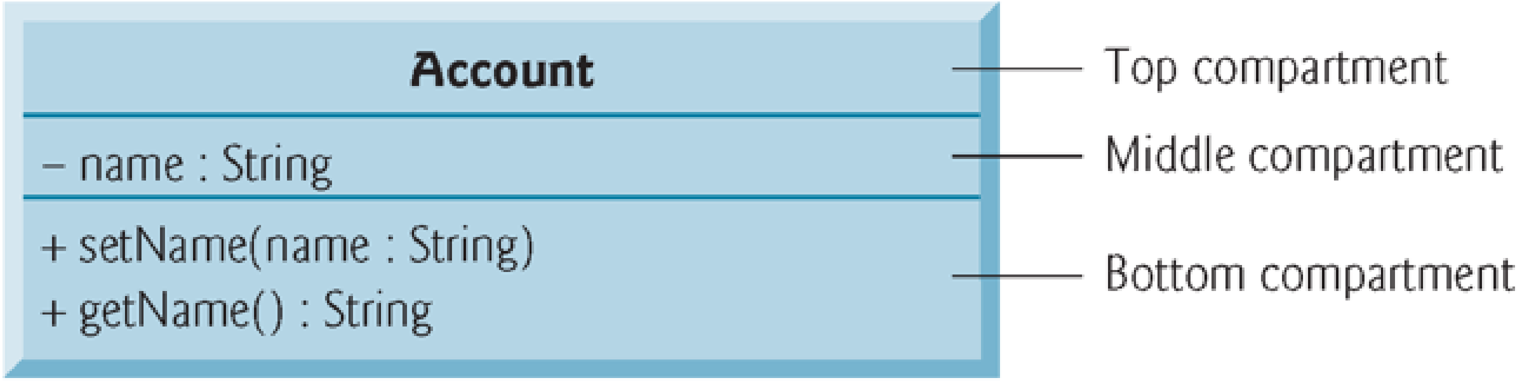
Bottom compartment

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

UML

Declaring instance variables `private` is known as data hiding or information hiding.





Account

– name : String

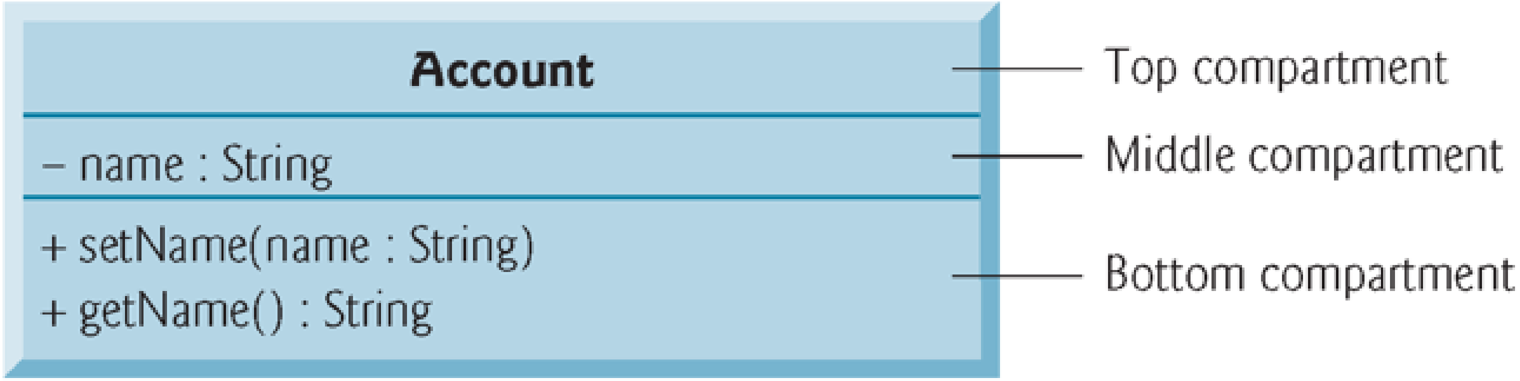
+ setName(name : String)
+ getName() : String

UML

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Precede each instance variable and method with an access specifier.

Generally, instance variables should be `private` and methods `public`.



Default and Explicit Initialization for Instance Variables

So what happened with `String`? Why was it set to `null`?

Local variables are not initialized by default.

Primitive-type instance variables are initialized by default

- instance variables of types `byte`, `char`, `short`, `int`, `long`, `float` and `double` are initialized to 0
- instance variables of type `boolean` are initialized to false

You can specify your own initial value for a primitive-type instance variable by assigning the variable a value in its declaration, as in

```
private int numberOfStudents = 10;
```

Primitive Types vs Reference Types

Java’s types are divided into primitive types and reference types.

Primitive types

```
boolean, byte, char, short, int,
long, float , double
```

All nonprimitive types are reference types.

Primitive Types vs Reference Types

A primitive-type variable can hold exactly one value of its declared type at a time.

Programs use variables of reference types (normally called references) to store the addresses of objects in the computer’s memory.

These variables are said to refer to an object in the program.

Primitive Types vs Reference Types

To call an object’s methods, you need a reference to the object.

If an object’s method requires additional data to perform its task, then you’d pass arguments in the method call.

Primitive-type variables do not refer to objects, so such variables cannot be used to invoke methods.

Primitive Types vs Reference Types

A variable of type `String` is a reference to the instance of the `String` object you instantiated when you created the variable.

Reference-type instance variables (such as those of type `String`), if not explicitly initialized, are initialized by default to the value `null`

`null`

represents a “reference to nothing.”

Constructors

Each class you declare can optionally provide a constructor with parameters that can be used to initialize an object of a class when the object is created.

Java requires a constructor call for every object that’s created.

If a class does not define constructors, the compiler provides a default constructor with no parameters, and the class’s instance variables are initialized to their default values.

Constructors

```
public class Account
{
    private String name;

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Class `Account` does not contain a constructor.

The default constructor is being used.

How do we know?

Constructors are special methods that share the name of class.

Let's add a constructor.

```
public class Account
{
    private String name;

    public Account(String name)
    {
        this.name = name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Class `Account` now contains a constructor.

How do we know?

There's a method in the class with the same name as the class

`Account`

What does the constructor do?

When the object is instantiated, the private instance variable `name` is set to the `String` parameter passed in the constructor.

Constructors

Now let's instantiate multiple objects in our program.

```
public class AccountDemo
{
    public static void main(String[] args)
    {
        Account sAccount = new Account("Superman");
        Account bAccount = new Account("Batman");

        System.out.printf("sAccount is named %s\n", sAccount.getName());
        System.out.printf("bAccount is named %s\n", bAccount.getName());
    }
}
```

```
sAccount is named Superman
bAccount is named Batman
```

Constructors Cannot Return Values

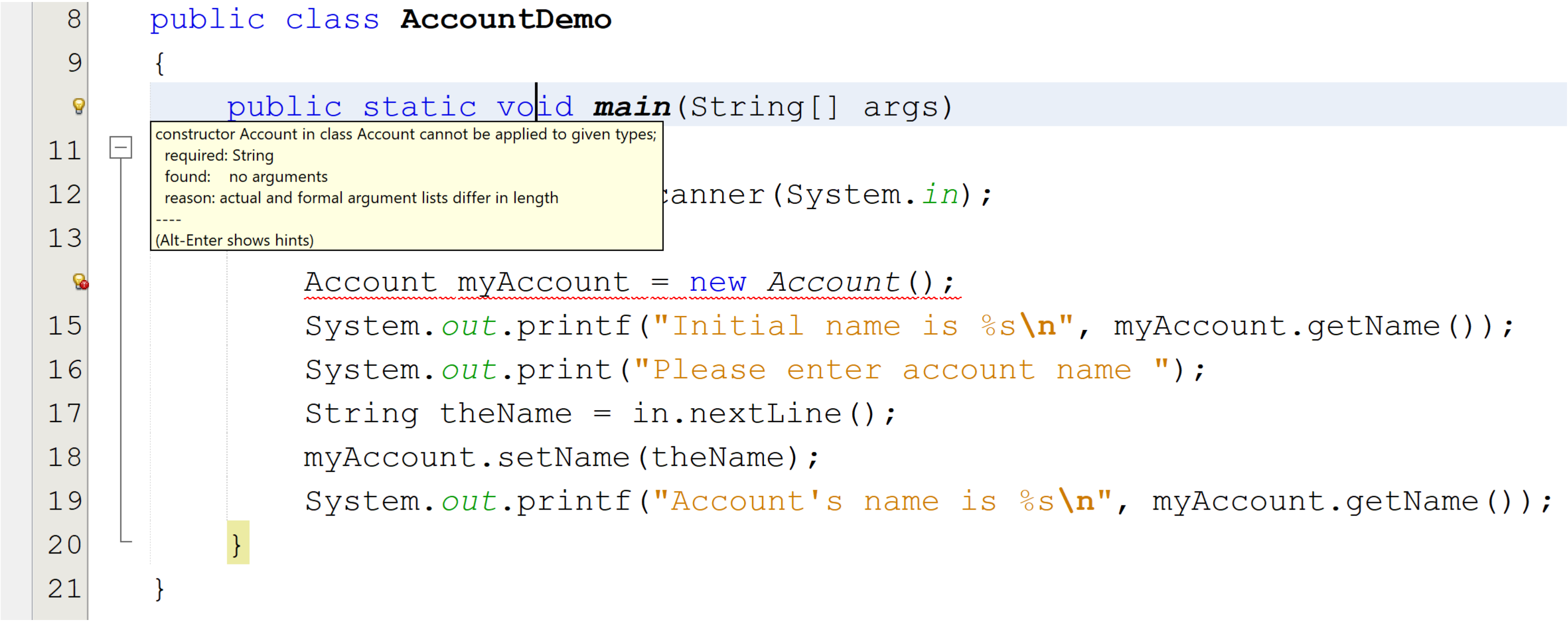
```
public class Account
{
    private String name;

    public Account(String name)
    {
        this.name = name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

Constructors can specify parameters but not return types.

This constructor is passed a `String` containing the name, but it does not return anything.

There’s no default constructor in a class that declares a constructor

If you declare a constructor for a class, the compiler will not create a default constructor for that class.



```
public class AccountDemo
{
    public static void main(String[] args)
                                        anner(System.in);

        Account myAccount = new Account();
        System.out.printf("Initial name is %s\n", myAccount.getName());
        System.out.print("Please enter account name ");
        String theName = in.nextLine();
        myAccount.setName(theName);
        System.out.printf("Account's name is %s\n", myAccount.getName());
    }
}
```

```
constructor Account in class Account cannot be applied to given types;
  required: String
  found:    no arguments
  reason: actual and formal argument lists differ in length
----
(Alt-Enter shows hints)
```

Constructors

Why use constructors?

If you do not provide a constructor, then your object will use the default initializations for the instance variables.

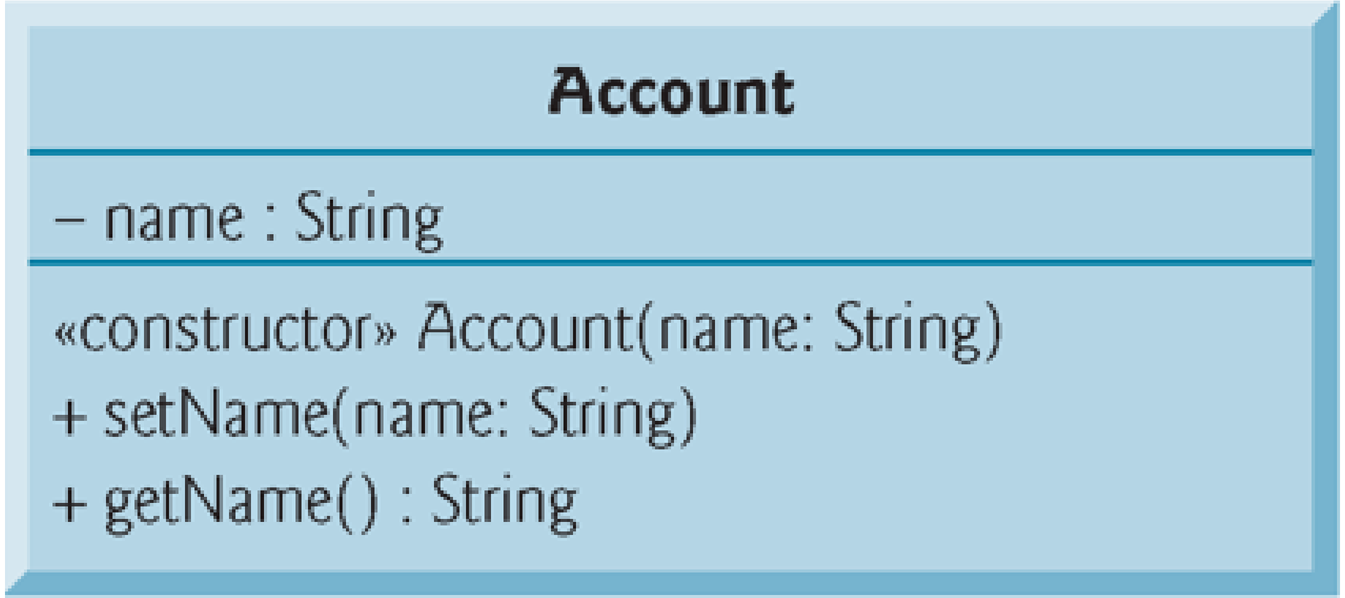
`0` for primitive and `null` for reference types

Use a constructor to start your object out with meaningful values.

Adding the Constructor to the class diagram

The constructor goes in the 3rd compartment of the class diagram

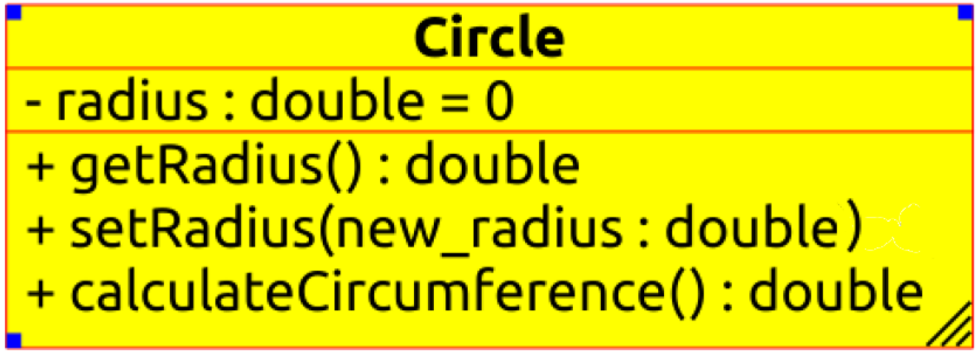
To distinguish a constructor from a class’s operations, place the word “constructor” between guillemets (« and ») before the constructor’s name.



Account

– name : String

«constructor» Account(name: String)
+ setName(name: String)
+ getName() : String

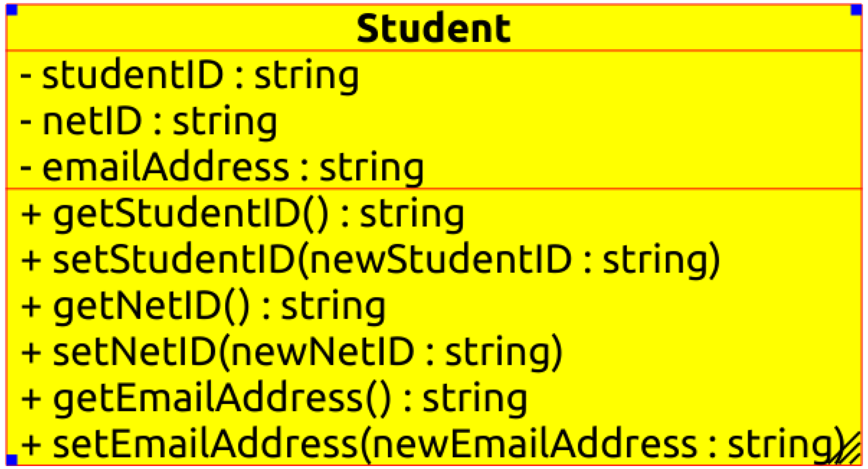


Circle

- radius : double = 0

+ getRadius() : double
+ setRadius(new_radius : double)
+ calculateCircumference() : double

```
public class Circle
{
    private double radius = 0;

    public double getRadius()
    {
    }
    public void setRadius(double new_radius)
    {
    }
    public double calculateCircumference()
    {
    }
}
```



Student

- studentID : string
- netID : string
- emailAddress : string

+ getStudentID() : string
+ setStudentID(newStudentID : string)
+ getNetID() : string
+ setNetID(newNetID : string)
+ getEmailAddress() : string
+ setEmailAddress(newEmailAddress : string)

```
public class Student
{
    private String studentID;
    private String netID;
    private String emailAddress;

    public String getStudentID()
    {
    }
    public void setStudentID(String newStudentID)
    {
    }
    public String getNetID()
    {
    }
}
```

```
public class PvsP
{
    public int x = 1;
    public char y = 'A';
    public double z = 2.3;

    private String a = "xyz";
    private float b = 1.2F;
    private long c = 123;
}
```

Which lines will compile?

```
PvsP QuizMe = new PvsP();
```

🟩 `QuizMe.x = 1;`

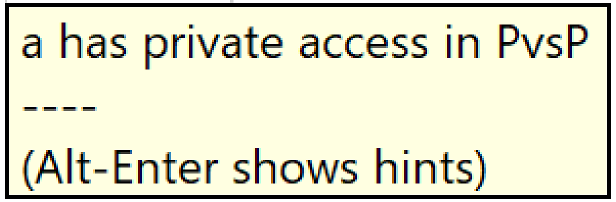
🟥 `QuizMe.a = "Quiz";`

🟩 `QuizMe.y = 'A';`

🟥 `QuizMe.b = 1.23F;`

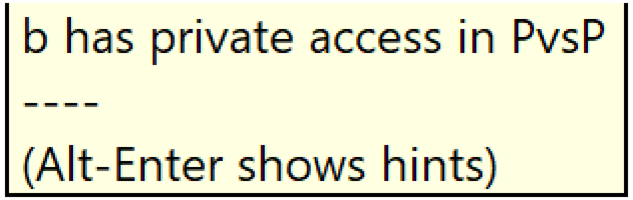
🟩 `QuizMe.z = 1.23;`

🟥 `QuizMe.c = 123;`



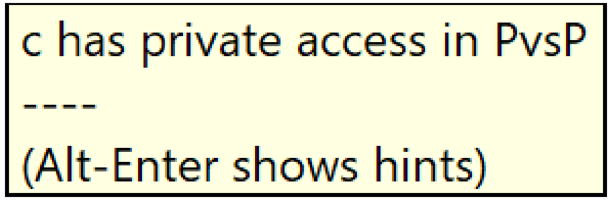
a has private access in PvsP

(Alt-Enter shows hints)



b has private access in PvsP

(Alt-Enter shows hints)



c has private access in PvsP

(Alt-Enter shows hints)

Encapsulation

You will begin to notice that objects tend to have mostly private data members and public member functions.

Why?

Take the example of the electronic devices that surround us every day.

They have simple interfaces that allows you to perform actions without know the details behind those actions.

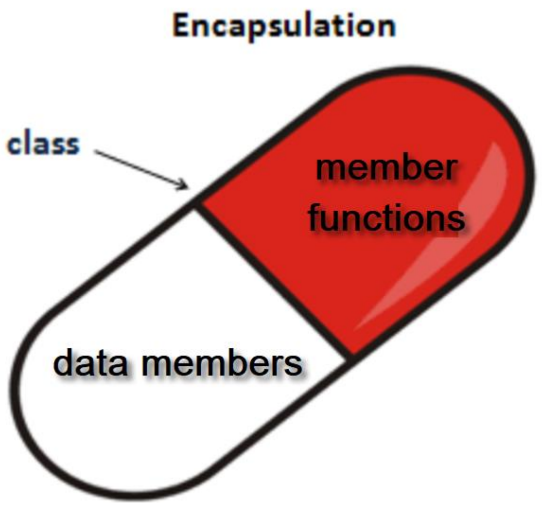
Encapsulation

The separation of interface and implementation is extremely useful because it allows us to use objects without understanding how they work.

This vastly reduces the complexity of using these objects and increases the number of objects we are capable of interacting with.

This same principle is applied to programming – separating implementation from interface.

Generally, data members are made private (hiding implementation details) and member functions are made public (giving the user an interface).



Encapsulation

Classes wrap attributes and member functions into objects created from those classes – an object's attributes and methods are intimately related.

Objects may communicate with one another, but they are not normally allowed to know how other objects are implemented.

Encapsulation is the technique of information hiding - implementation details are hidden within the objects themselves.

Encapsulation

Benefits of Encapsulation

Encapsulated classes are easier to use and reduce the complexity of programs.

Encapsulated classes help protect your data and prevent misuse

Encapsulated classes are easier to debug

Encapsulation

Access Functions

Getter

A method that returns the value of a private variable

Setter

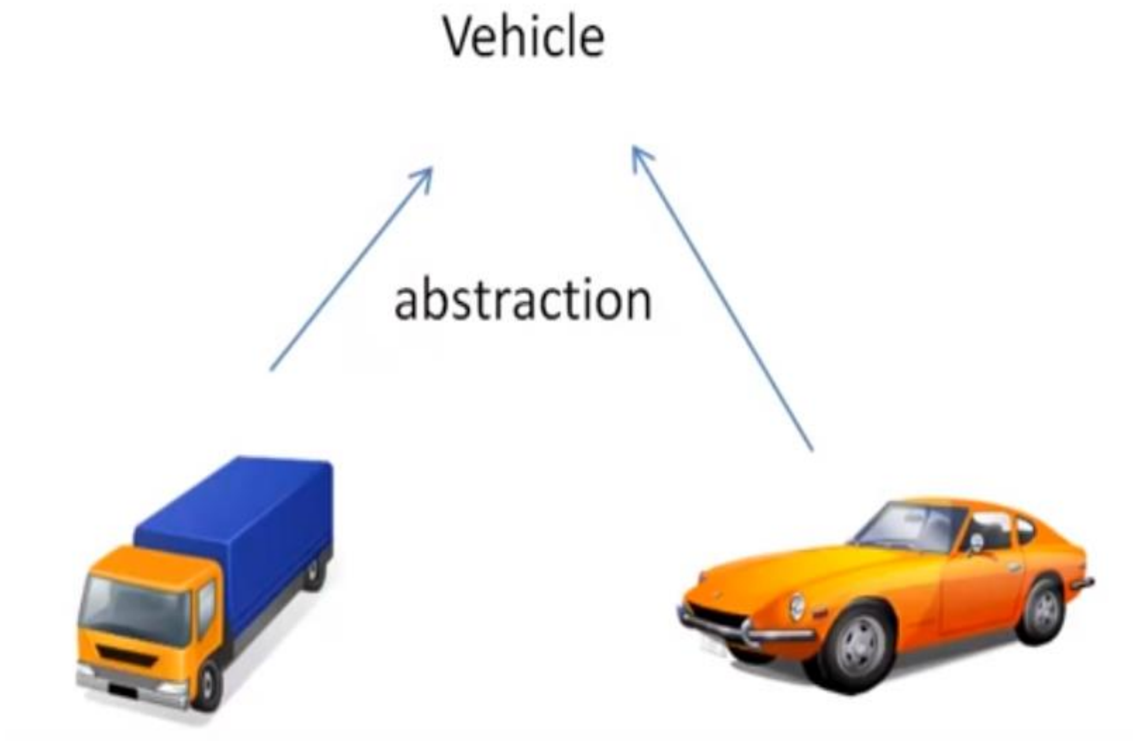
A method that changes the value of a private variable

Abstraction

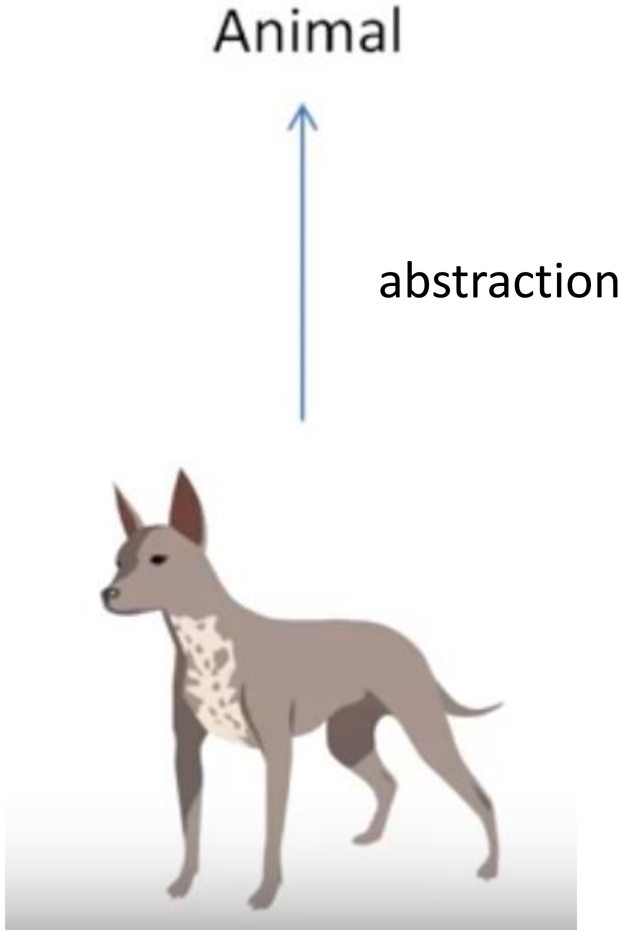
Abstraction is the concept of describing something in simpler terms, i.e abstracting away the details, in order to focus on what is important.

Abstraction is used to reduce complexity and allow efficient design and implementation of complex software systems.

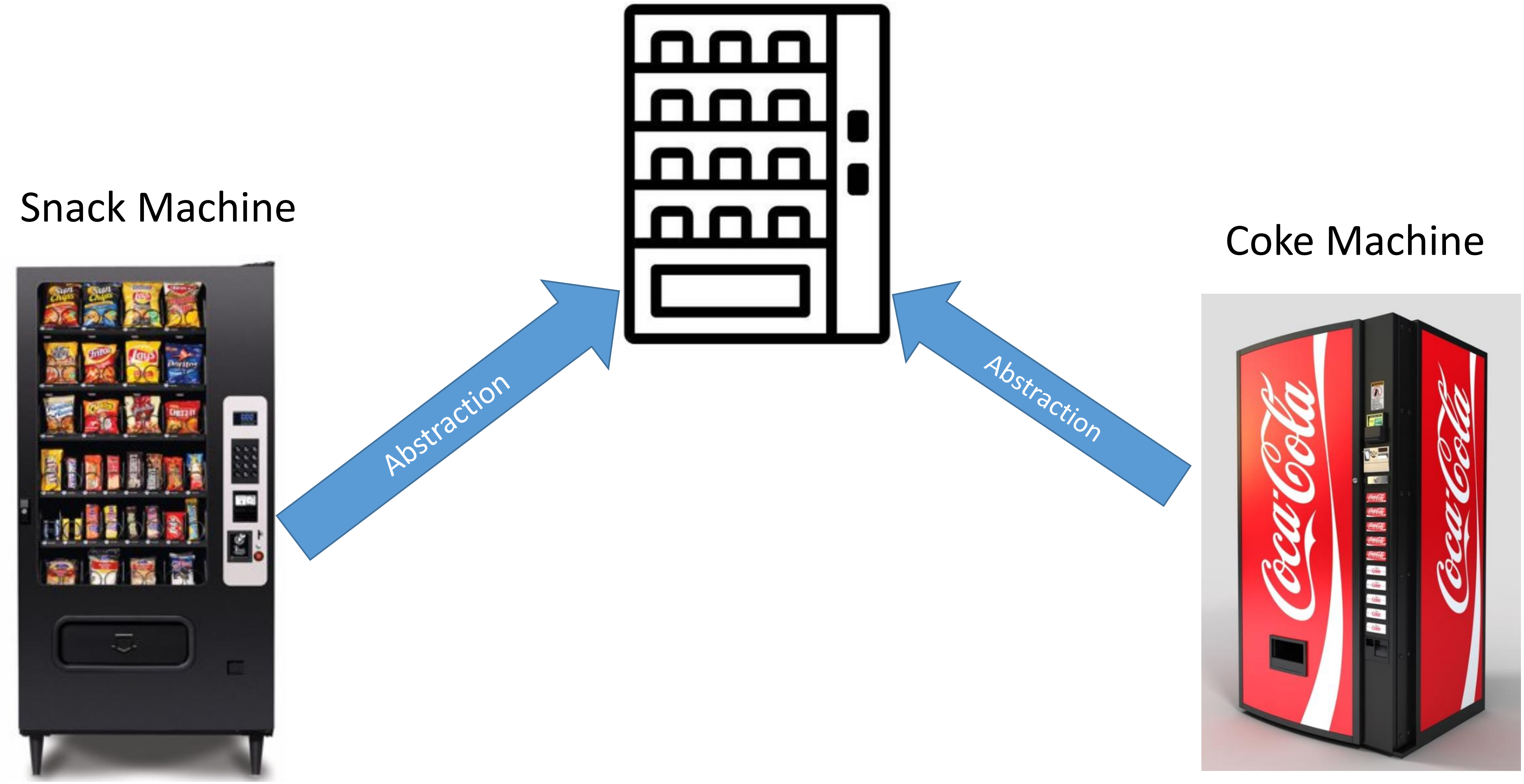
Abstraction is the act of representing essential features without including the background details or explanations.



Vehicle is an abstraction of truck and car.



Animal is an abstraction of dog.



Vending Machine

Snack Machine

Abstraction

Abstraction

Coke Machine

What is this?

A tree?

How do you know?

We are aware of the abstract concept of a tree and are able to recognize this picture as an instantiation of our abstract version of "tree".

So do we make assumptions about this tree even though we've never met this tree?

Yes

This tree is made up of leaves, branches, a trunk and roots.

It needs water and sunlight to live.

Object Relationships

We are going to explore the relationship between objects based on what we know about them.

There are many different kinds of relationships two objects may have in real-life and we use specific “relation type” words to describe these relationships.

Object Relationships

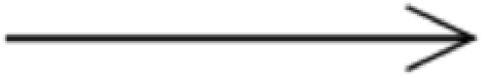
A square "is a" shape

A car "has a" steering wheel

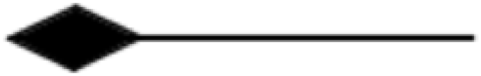
UML Relationships

There are many methods of showing relationships between classes.
We are going to focus on four specific relationships.

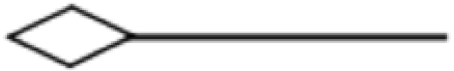
Association



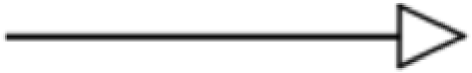
Composition



Aggregation

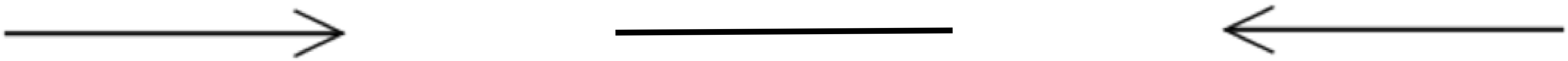


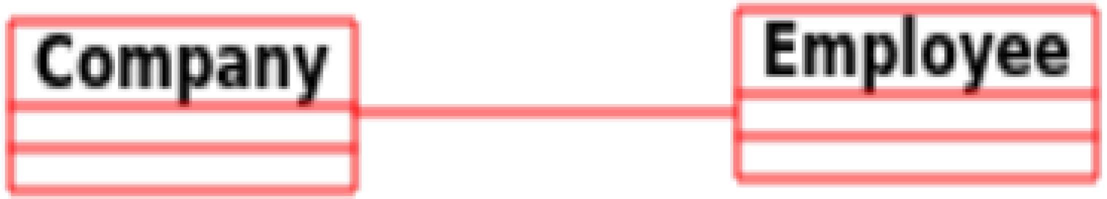
Inheritance



UML Relationships

Association





- Represents the "has a" relationship
- a linkage between two classes
- shows that classes are aware of each other and their relationship
- uni-directional or bi-directional

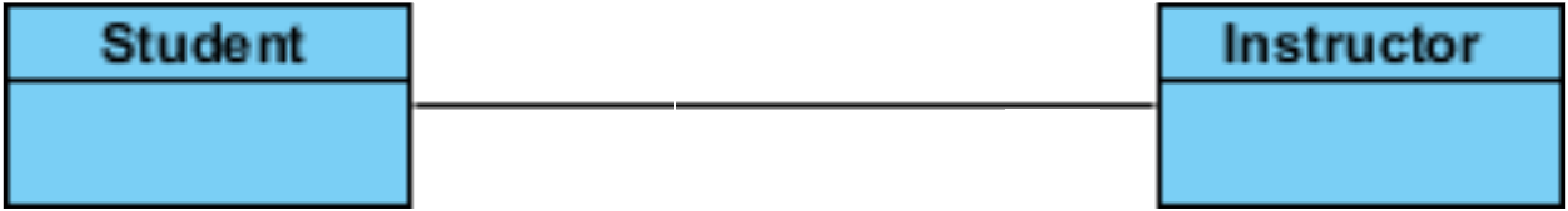
Object Relationships - Association

A Student can be associated with an Instructor

Student
Instructor

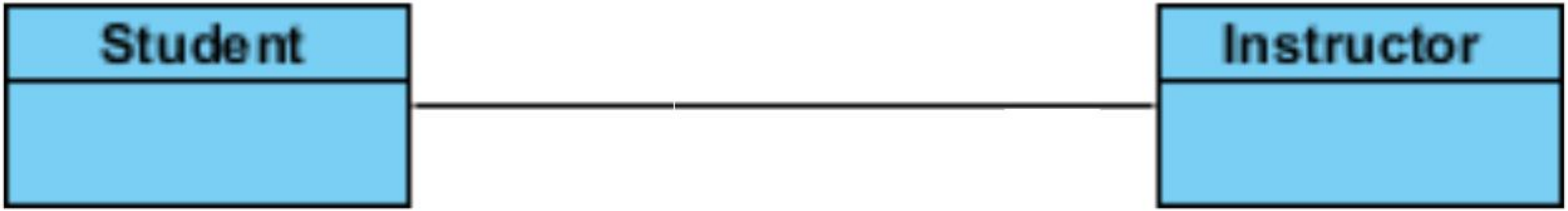
Object Relationships - Association

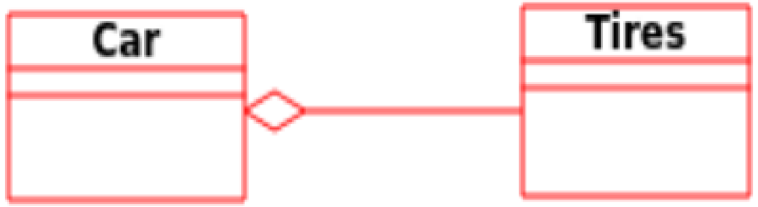
A single Student can be associated with multiple Instructors



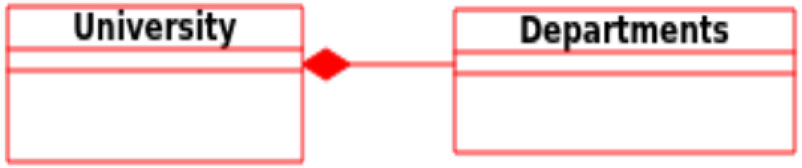
Object Relationships - Association

Multiple Students can be associated with multiple Instructors

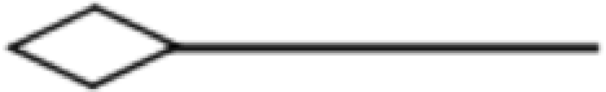




UML Relationships

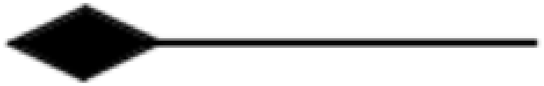


Aggregation



- Special type of association
- Represents the "has a" /"whole-part" relationship.
- Describes when a class (the whole) is composed of/has other classes (the parts)
- Diamond on "whole" side

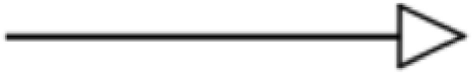
Composition

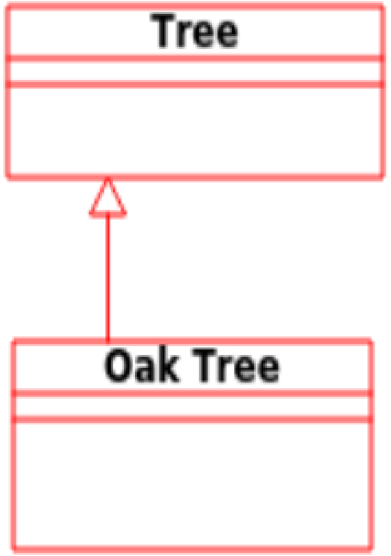


- An association that represents a very strong aggregation
- Represents the "has a" /"whole-part" relationship.
- Describes when a class (the whole) is composed of/has other classes (the parts) BUT the parts cannot exist without the whole.
- Diamond on "whole" side

UML Relationships

Inheritance





- Represents the "is a" relationship

- Shows the relationship between a super class/base class and a derived/subclass.

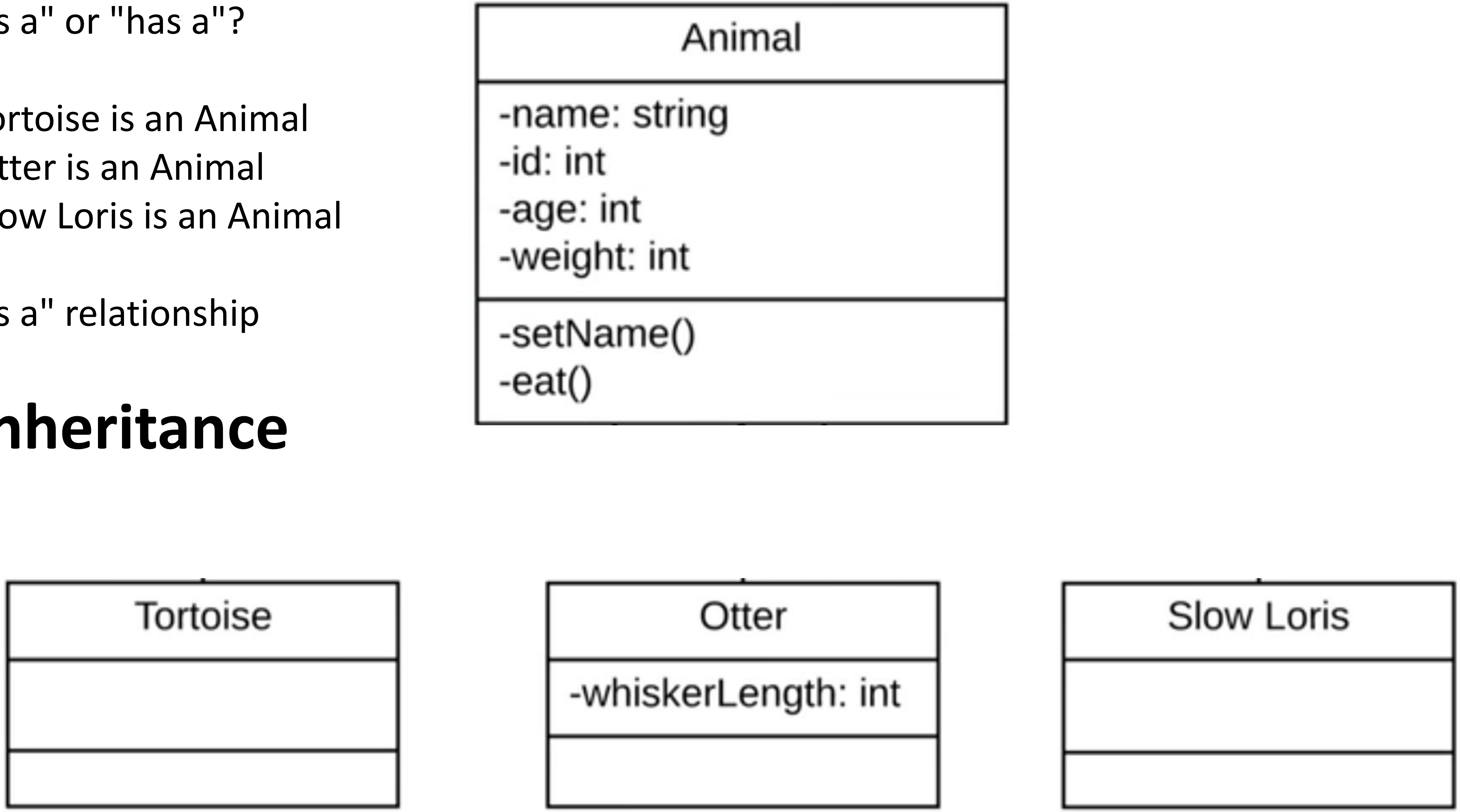
- Arrow is on the side of the base class

"is a" or "has a"?

Tortoise is an Animal
Otter is an Animal
Slow Loris is an Animal

"is a" relationship

Inheritance



Animal

-name: string
-id: int
-age: int
-weight: int

-setName()
-eat()

Tortoise

Otter

-whiskerLength: int

Slow Loris

"is a" or "has a"?

Otter is a Sea Urchin?
Otter has a Sea Urchin?

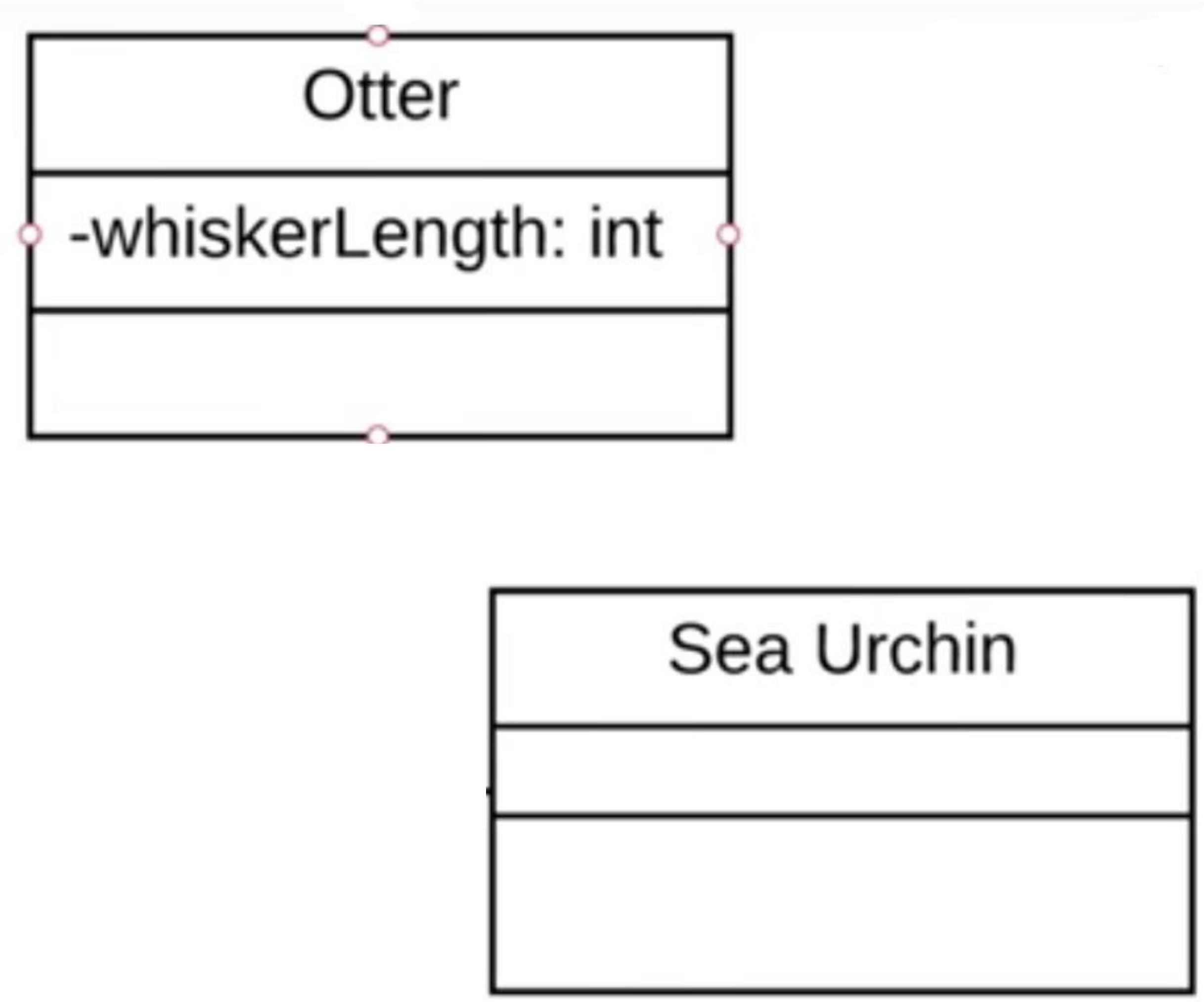
"has a" relationship

association/composition/aggregation?

"whole/part" relationship?

Otter is part of Sea Urchin?
Sea Urchin is part of Otter?

not "whole/part"



Association

"is a" or "has a"?

Creep is a Tortoise?
Creep has a Tortoise?

"has a" relationship

association/composition/aggregation?

"whole/part" relationship?

Creep is part of Tortoise?
Tortoise is part of Creep?

Yes – Creep is "whole" and Tortoise is "part"

Can the Creep exist without the Tortoise?

Yes

Aggregration

Creep
Tortoise

"is a" or "has a"?

Lobby is a Visitor Center? Bathroom is a Visitor Center?
Visitor Center is a lobby or bathroom?

Lobby/Bathroom has a Visitor Center?
Visitor Center has a Lobby and a Bathroom?

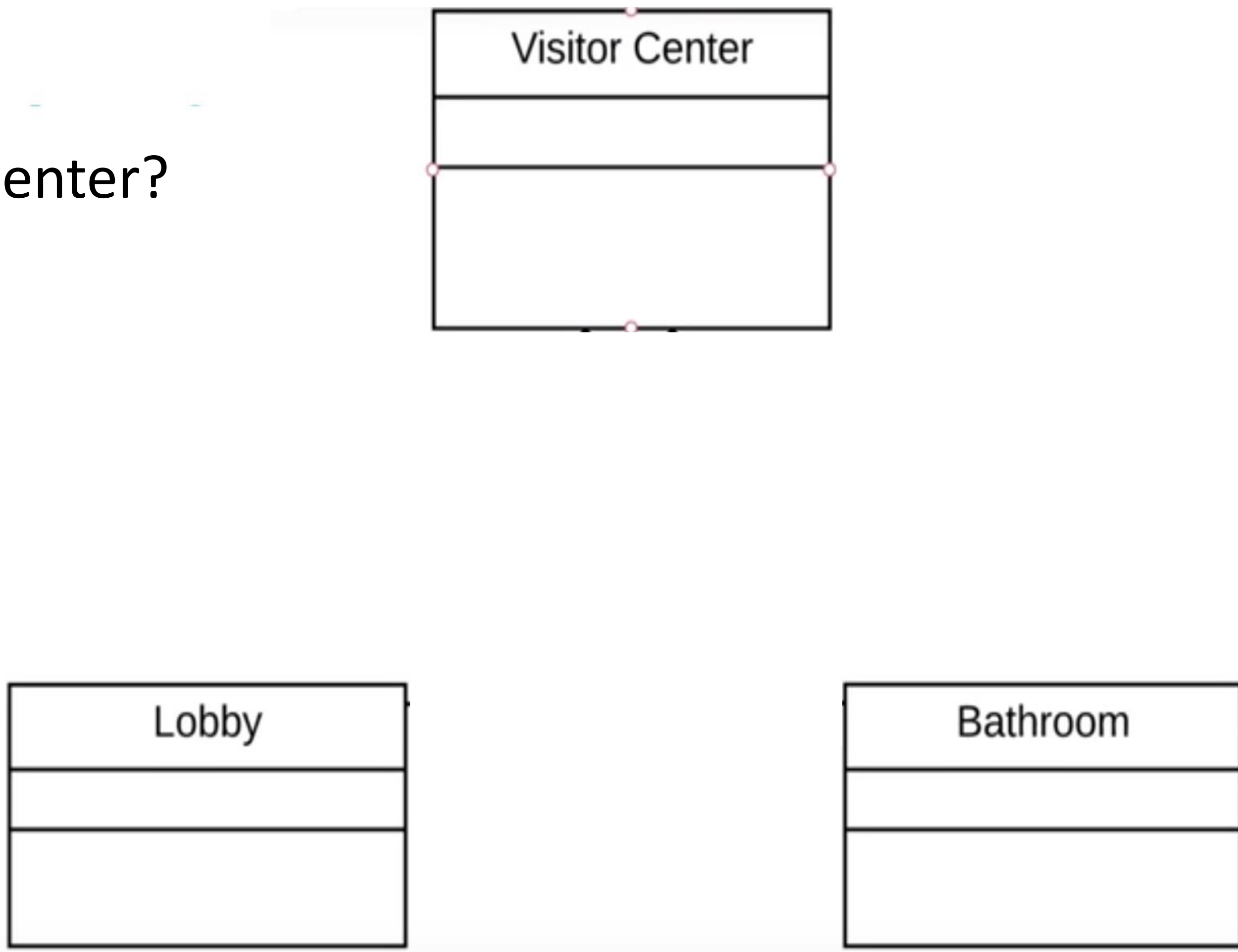
"has a" relationship

association/composition/aggregation?

"whole/part" relationship?

Visitor Center is part of Bathroom/Lobby?
Bathroom/Lobby is part of Visitor Center?

Yes – Visitor Center is "whole" and Bathroom/Lobby is "part"



Can the "parts" – Bathroom and Lobby – exist without the "whole" - Visitor Center?

Composition

Separating Interface from Implementation

The client code actually should only know 3 things about a class

- what instance methods to call
- what arguments to provide to each method
- what return type to expect from each method

The client code does not need to know how those methods are implemented.

Separating Interface from Implementation

When the client code does know how a class is implemented, then the programmer might write client code based on the class's implementation details.

Ideally, if the class's implementation changes, the class's clients should not have to change.

Hiding the class's implementation details makes it easier to change the class's implementation while minimizing, and hopefully, eliminating changes to the client code.

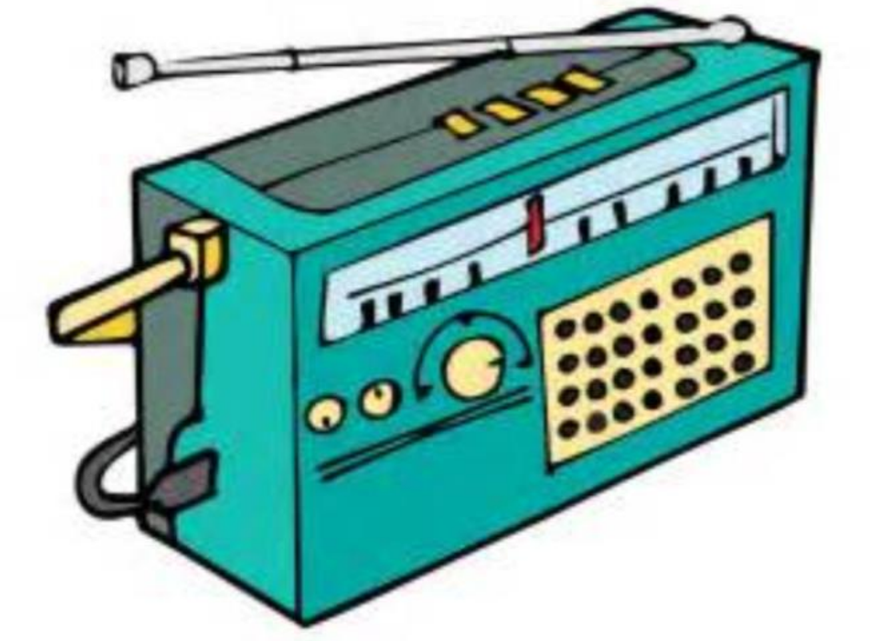
Interface of a Class

We are all familiar with a radio, and, in general, the controls that allow us to perform a limited set of operations on the radio – changing the station, adjusting the volume and choosing between AM and FM.

Some radios use dials, some have push buttons and some support voice commands.

These controls serve as the interface between the radio's users and the internal components.

The interface specifies what operations a radio permits users to perform but does not specify how the operations are implemented inside the radio.



Interface of a Class

The interface of a class describes *what* services a class's clients can use and how to *request* those services, but not *how* the class carries out the services.

A class's `public` interface consists of the class's `public` member methods which can also be known as the class's `public` services.

We can specify a class's interface by writing a class diagram that lists only the class's methods prototypes and the class's instance variables.

```
public class Account
{
    private String name;

    public Account(String name)
    {
        this.name = name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

```
public class Account
{
    private String name;
    private double balance;

    public Account(String name, double balance)
    {
        this.name = name;
        if (balance > 0.0)
        {
            this.balance = balance;
        }
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

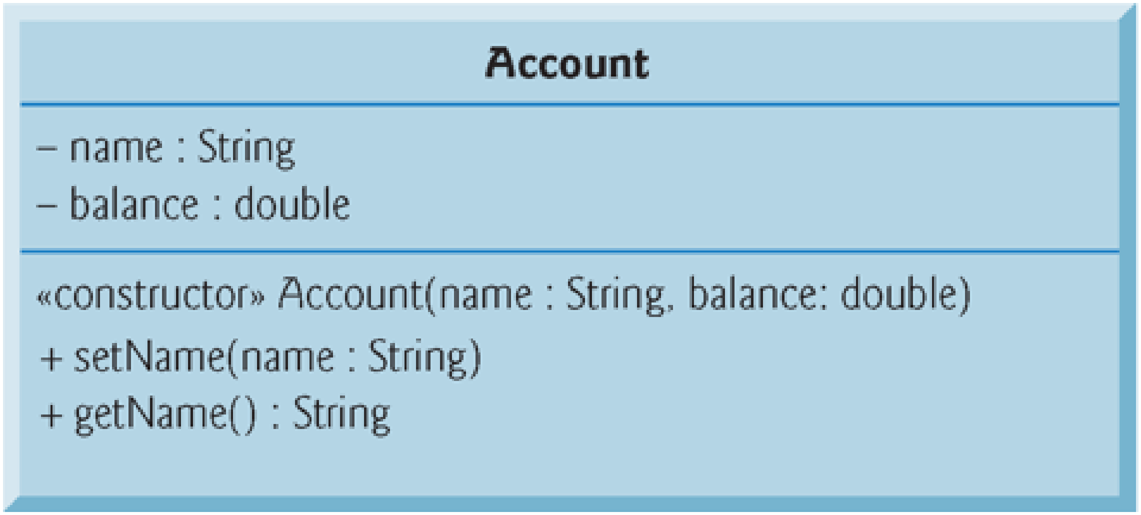
Added `balance`

Validates `balance`

```
public class Account
{
    private String name;
    private double balance;

    public Account(String name, double balance)
    {
        this.name = name;
        if (balance > 0.0)
        {
            this.balance = balance;
        }
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

`balance` is now a variable available to an instance method of class `Account`

The constructor has `balance` as a parameter.



Account

– name : String
– balance : double

«constructor» Account(name : String, balance: double)
+ setName(name : String)
+ getName() : String

```
public class Account
{
    private String name;
    private double balance;

    public Account(String name, double balance)
    {
        this.name = name;
        if (balance > 0.0)
        {
            this.balance = balance;
        }
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

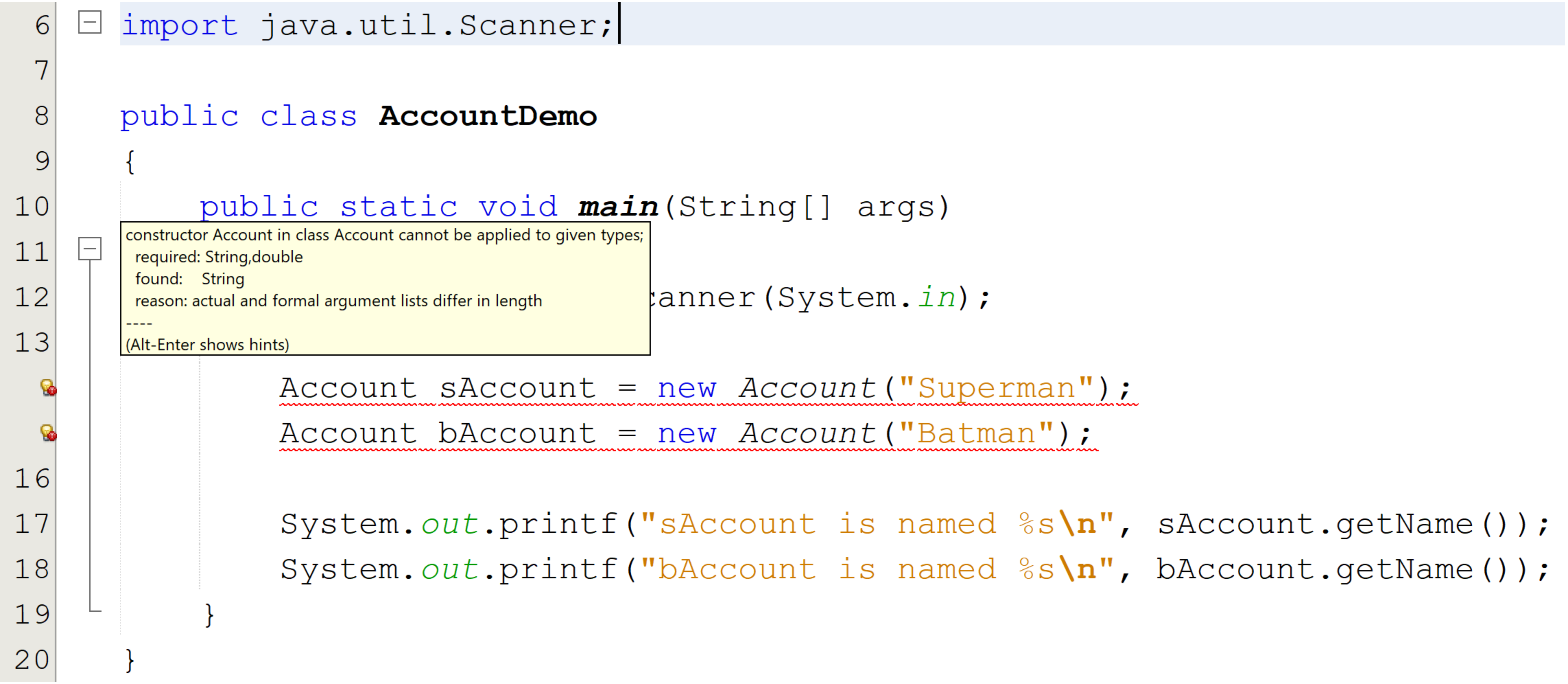
Constructor validates that the passed in `balance` is greater than zero.

If it is, the object's `balance` instance variable is set to the passed in balance; else, `balance` is set to what value?

0.0

Why?

Primitive instance variables are initialized to 0 and type `double` is set to 0.0



```
import java.util.Scanner;

public class AccountDemo
{
    public static void main(String[] args)

                          anner(System.in);

            Account sAccount = new Account("Superman");
            Account bAccount = new Account("Batman");

            System.out.printf("sAccount is named %s\n", sAccount.getName());
            System.out.printf("bAccount is named %s\n", bAccount.getName());
    }
}
```

```
constructor Account in class Account cannot be applied to given types;
  required: String,double
  found:    String
  reason: actual and formal argument lists differ in length
----
(Alt-Enter shows hints)
```

We started with the default constructor.

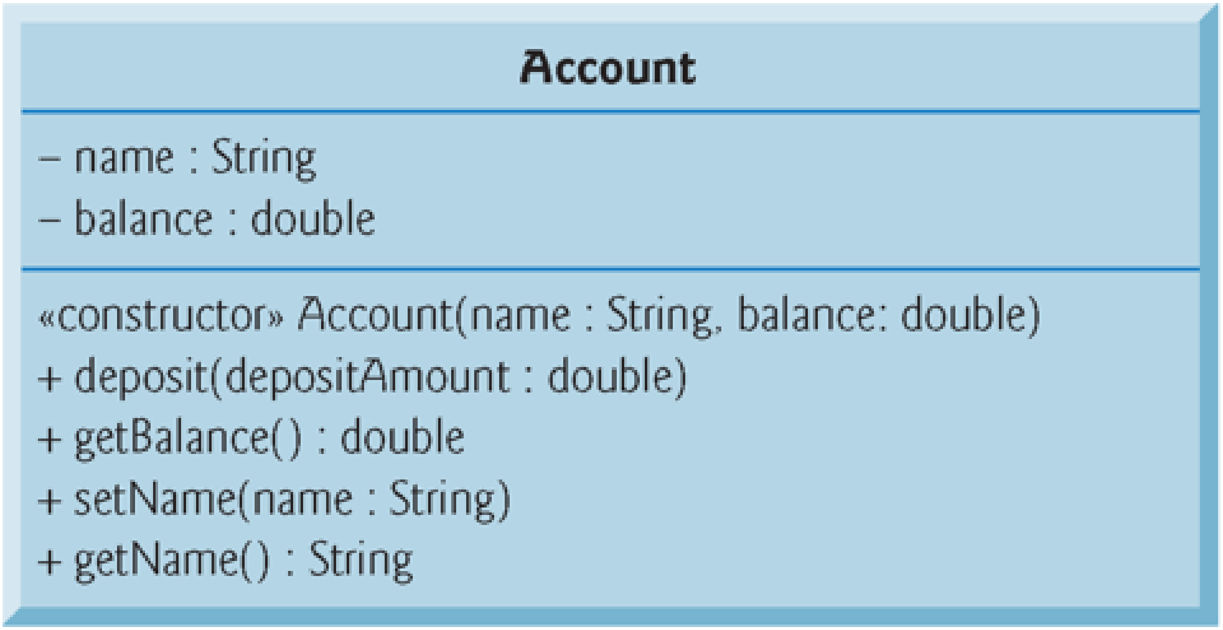
We then created our own constructor with one parameter to set `name` to the passed in value.

Then, we add a passed in `balance` to the constructor.

```
import java.util.Scanner;

public class AccountDemo
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        Account sAccount = new Account("Superman", -7);
        Account bAccount = new Account("Batman", 1000000);

        System.out.printf("sAccount is named %s\n", sAccount.getName());
        System.out.printf("bAccount is named %s\n", bAccount.getName());
    }
}
```

Adding a new instance methods `deposit` and `getBalance`

```
public void deposit(double depositAmount)
{
   if (depositAmount > 0.0)
   {
      balance += depositAmount;
   }
}
public double getBalance()
{
   return balance;
}
```



Account

– name : String
– balance : double

«constructor» Account(name : String, balance: double)
+ deposit(depositAmount : double)
+ getBalance() : double
+ setName(name : String)
+ getName() : String

```
sAccount is named Superman and has a balance of $0.00
bAccount is named Batman and has a balance of $1000000.00
```

```
package accountdemo;

import java.util.Scanner;

public class AccountDemo
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        Account sAccount = new Account("Superman", -7);
        Account bAccount = new Account("Batman", 1000000);

        System.out.printf("sAccount is named %s and has a balance of $%.2f\n",
            sAccount.getName(), sAccount.getBalance());
        System.out.printf("bAccount is named %s and has a balance of $%.2f\n",
            bAccount.getName(), bAccount.getBalance());
    }
}
```

```
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        Account sAccount = new Account("Superman", -7);
        Account bAccount = new Account("Batman", 1000000);

        System.out.printf("sAccount is named %s and has a balance of $%.2f\n",
            sAccount.getName(), sAccount.getBalance());
        System.out.printf("bAccount is named %s and has a balance of $%.2f\n",
            bAccount.getName(), bAccount.getBalance());

        System.out.printf("How much money should Batman give Superman? ");
        double batLoan = in.nextDouble();
        sAccount.deposit(batLoan);

        System.out.printf("Superman now has $%.2f in his account\n",
            sAccount.getBalance());
    }
```

```
sAccount is named Superman and has a balance of $0.00
bAccount is named Batman and has a balance of $1000000.00
How much money should Batman give Superman? .01
Superman now has $0.01 in his account
```

```
public boolean withdraw(double withdrawalAmount)
{
    if (withdrawalAmount <= balance)
    {
        balance -= withdrawalAmount;
        return true;
    }
    else
    {
        return false;
    }
}
```

```
public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);

    Account sAccount = new Account("Superman", -7);
    Account bAccount = new Account("Batman", 1000000);

    System.out.printf("sAccount is named %s and has a balance of $%.2f\n",
        sAccount.getName(), sAccount.getBalance());
    System.out.printf("bAccount is named %s and has a balance of $%.2f\n",
        bAccount.getName(), bAccount.getBalance());

    System.out.printf("How much money should Batman give Superman? ");
    double batLoan = in.nextDouble();
    if (bAccount.withdraw(batLoan))
    {
        sAccount.deposit(batLoan);

        System.out.printf("Superman now has $%.2f in his account",
            sAccount.getBalance());
        System.out.printf(" and Batman has $%.2f in his account.\n",
            bAccount.getBalance());
    }
    else
    {
        System.out.printf("Batman does not have that much money!!\n");
    }
}
```

```
sAccount is named Superman and has a balance of $0.00
bAccount is named Batman and has a balance of $1000000.00
How much money should Batman give Superman? .01
Superman now has $0.01 in his account and Batman has $999999.99 in his account.
```

```
sAccount is named Superman and has a balance of $0.00
bAccount is named Batman and has a balance of $1000000.00
How much money should Batman give Superman? 1000000
Superman now has $1000000.00 in his account and Batman has $0.00 in his account.
```

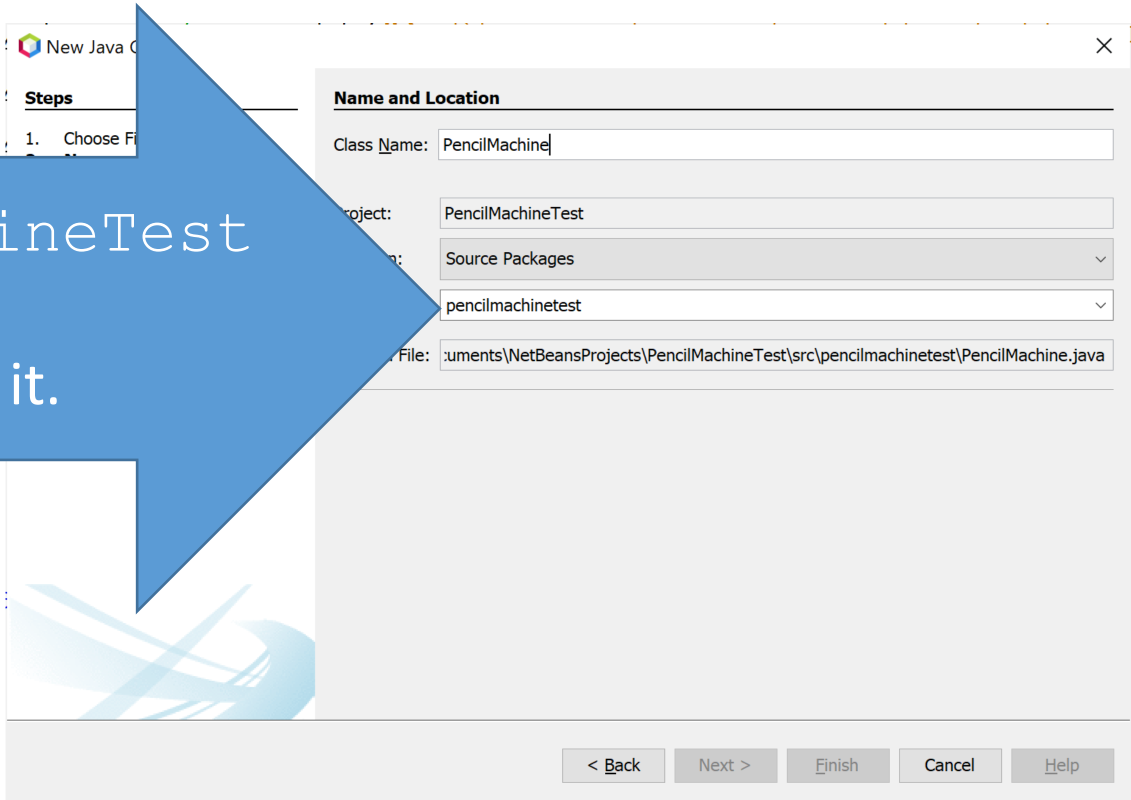
```
sAccount is named Superman and has a balance of $0.00
bAccount is named Batman and has a balance of $1000000.00
How much money should Batman give Superman? 1000000.01
Batman does not have that much money!!
```

Converting our Pencil Machine to an Object

What would a Pencil Machine object look like?

```
public class PencilMachine
{



}
```

Create a new project called `PencilMachineTest` and copy your code from `Code2_xxxxxxxxxx.java` into it.



New Java C

Steps

1. Choose Fi

Name and Location

Class Name: PencilMachine

roject: PencilMachineTest

n: Source Packages

pencilmachinetest

File: :uments\NetBeansProjects\PencilMachineTest\src\pencilmachinetest\PencilMachine.java

Converting our Pencil Machine to an Object

```
package pencilmachinetest;

public class PencilMachine
{
   public enum ACTION
   {
       DISPENSECHANGE, NOINVENTORY, NOCHANGE, INSUFFICIENTFUNDS, EXACTPAYMENT
   }

   public final int PENCIL_PRICE;
   public int inventoryLevel;
   public int changeLevel;
   public int changeToBeDispensed;

}
```

Including the `enum` in our class and making it `public` allows all of our classes, including `pencilmachinetest`, to see it.

We'll make all of our instance variables `public` for now.

Converting our Pencil Machine to an Object

If we use the default constructor to instantiate our Pencil Machine object

```
PencilMachine PM = new PencilMachine();
```

```
0. No pencils for me today
1. Purchase pencils
2. Check inventory level
3. Check change level

Choice : 2

The current inventory level is 0



Choice : 3

The current change level is $0.00
```

Primitive instance variables are initialized to 0 when objects are instantiated.

Converting our Pencil Machine to an Object

Let's build our own constructor so that change level and inventory level can start at the values we read from our file.

```
public PencilMachine(int startingChange, int startingInventory,
                     int PencilPrice)
{
    inventoryLevel = startingInventory;
    changeLevel = startingChange;
    PENCIL_PRICE = PencilPrice;
}
```

```
The current inventory level is 100

The current change level is $5.00
```

What will be the value of

`changeToBeDispensed`?

Do we need to set it to a value?

No, because it will be set to 0.

Converting our Pencil Machine to an Object

What would the constructor look like if we used the same names for our variables?

```
public PencilMachine(int changeLevel, int inventoryLevel, int PENCIL_PRICE)
{
    this.inventoryLevel = inventoryLevel;
    this.changeLevel = changeLevel;
    this.PENCIL_PRICE = PENCIL_PRICE;
}
```

Converting our Pencil Machine to an Object

```
public static void main(String[] args)
{
    /* Read values from file */

    PencilMachine PM = new PencilMachine(change

    int payment = 0;
    int quantity = 0;
    int menu_choice = 0;

    PencilMachine.ACTION action;
```

PM

`PencilMachine` is the name of the class where the `enum` lives

Converting our Pencil Machine to an Object

```
switch (menu_choice)
{
   case 0 :
      break;
   case 1 :
      if (inventoryLevel != 0)  if (PM.inventoryLevel != 0)
```

`inventoryLevel` is no longer a variable local to `main()`

It is an instance variable now.

How do we get the inventory level?

Converting our Pencil Machine to an Object

```
do
{
    quantity = in.nextInt();

    if (quantity < 1 || quantity > inventoryLevel)
    {
        System.out.printf("Cannot ... Please ... ");
    }
}
while (quantity < 1 || quantity > inventoryLevel);
```

Converting our Pencil Machine to an Object

```
do
{
    quantity = in.nextInt();

    if (quantity < 1 || quantity > PM.inventoryLevel)
    {
        System.out.printf("Cannot ... Please ... ");
    }
}
while (quantity < 1 || quantity > PM.inventoryLevel);
```

Converting our Pencil Machine to an Object

```
case 2 :
    System.out.printf("\nThe current inventory level is %d\n", inventoryLevel);
    break;
case 3 :
    System.out.printf("\nThe current change level is %s\n", displayMoney(changeLevel));
    break;


case 2 :
    System.out.printf("\nThe current inventory level is %d\n", PM.inventoryLevel);
    break;
case 3 :
    System.out.printf("\nThe current change level is %s\n", displayMoney(PM.changeLevel));
    break;
```

Converting our Pencil Machine to an Object

```
System.out.printf("\nA pencil costs %s",
                   displayMoney(PENCIL_PRICE));
```

```
System.out.printf("\nA pencil costs %s",
                   displayMoney(PM.PENCIL_PRICE));
```

Converting our Pencil Machine to an Object

```
action = buyPencils(PENCIL_PRICE, payment, quantity,
                    Levels, change);
```

```
action = PM.buyPencils(payment, quantity);
```

The Pencil Machine knows how much a pencil costs, the current inventory level and change level and we added a new field to the object to know the amount of change to be dispensed.

Converting our Pencil Machine to an Object

We originally set up `PencilMachine` to have an integer to hold the amount of changed needed after any sale.

```
public int changeToBeDispensed;
```

This `int` would need to be formatted using `displayMoney()` before being displayed.

```
displayMoney(PM.changeToBeDispensed)
```

Is there any reason to **STORE** the amount of change to be dispensed as an integer?

Are we going to do calculations with it?

Converting our Pencil Machine to an Object

No. This variable is only used for displaying the amount of change dispensed.

So, how about we store it as the formatted version in a `String`?

```
public int changeToBeDispensed;
public String changeToBeDispensed;
```

```
changeToBeDispensed = displayMoney(payment - (PENCIL_PRICE * quantity));
```

Converting our Pencil Machine to an Object

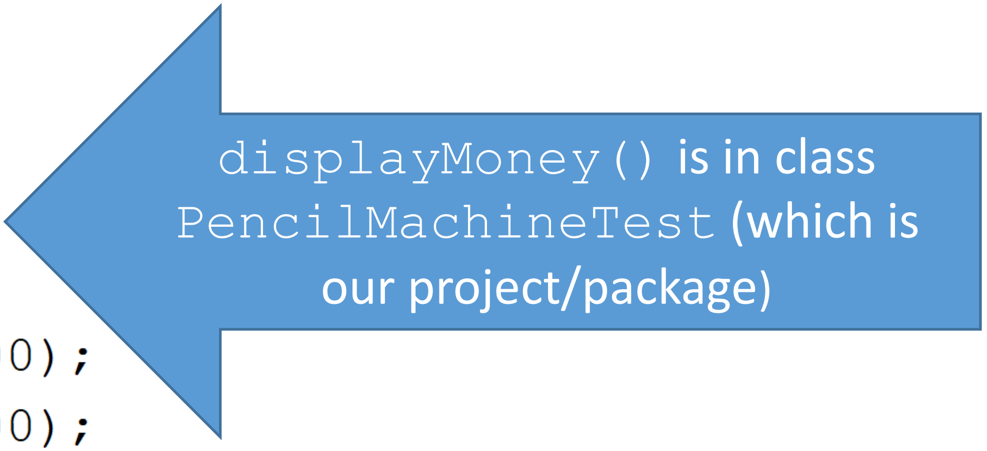
```
public String changeToBeDispensed;
```

```
changeToBeDispensed = displayMoney(payment - (PENCIL_PRICE * quantity));
```

```
error: cannot find symbol
                changeToBeDispensed = displayMoney(payment -
(PENCIL_PRICE * quantity));
  symbol:   method displayMoney(int)
  location: class PencilMachine
1 error
BUILD FAILED (total time: 1 second)
```

Converting our Pencil Machine to an Object

```
public class PencilMachineTest
{
    public static String displayMoney(int amount)
    {
        String dollars = String.valueOf(amount/100);
        String cents = String.valueOf(amount % 100);
        return "$" + dollars + "." + (cents.length() == 1 ? "0" : "") + cents;
    }
```



`displayMoney()` is in class `PencilMachineTest` (which is our project/package)

Converting our Pencil Machine to an Object

```
public class PencilMachineTest
{
   public static String displayMoney




}
```

```
public class PencilMachine
{
   PencilMachineTest.displayMoney()




}
```

package pencilmachinetest

Converting our Pencil Machine to an Object

```
public int changeToBeDispensed;
```

In the `PencilMachine` class

```
public String changeToBeDispensed;
```

~~changeToBeDispensed = displayMoney(payment - (PENCIL_PRICE * quantity));~~

```
changeToBeDispensed = PencilMachineTest.displayMoney(payment -
                            (PENCIL_PRICE * quantity));
```

```
--------------------------------------------------------------------------------
```

```
System.out.printf("Here's your pencils and your change of %s\n",
                         PM.changeToBeDispensed);
```

Converting our Pencil Machine to an Object

If we moved `displayMoney()`

from `PencilMachineTest`

into our `PencilMachine`

then we could just use it like any other method in our object.

Converting our Pencil Machine to an Object

Just because we can, does not mean that we should.

Separation of interface and implementation

Formatting the money values is an interface detail; therefore, our object should NOT be responsible for that process.

Converting our Pencil Machine to an Object

By making this change, we are causing our object to be dependent on code that is outside of it.

If we wanted to use our PencilMachine with a different interface, then we would need to make changes.

Negates encapsulation

Converting our Pencil Machine to an Object

Should we put our Pencil menu or `displayMoney()` inside the Pencil Machine object?

Separation of interface and implementation

Displaying the menu and formatting the money – are those part of the interface or the implementation?

Interface so leave them out of the object.

Converting our Pencil Machine to an Object

Could our menu method become its own class?

- Menus in different formats
- Read the menu contents from a file
- Different languages

Could our money formatter become its own class?

- Different currencies
- Exchange rates