Object Oriented Programming

6094-822-06

Content

Object Oriented Programming

Written by
Rony Keren
Internet Team
John-Bryce

JOHN BRYCE
Leading in IT Education
a matrix company

Problems with Linear Programming

JOHN BRYCE
Leading in IT Education
a matrix company

Problematic maintainability

- Applications are not structured in a standard way
- One change might break the chain
- Technical terms rather than real-world terms

High development costs

- Hard to reuse code
- Good code must be copied
- Code review is problematic
- Hard to debug

What Is OOP

JOHN BRYCE
Leading in IT Education
a matrix company

"a programming language that enables the programmer to associate a set of procedures with each type of data structure" *(hyper dictionary)*

Which means that:
The program is a sequence of function called by each other and located in different well defined entities.

Object Oriented Goals

JOHN BRYCE
Leading in IT Education
a matrix company

- Reuse
- Maintainability
- Ease of development
- Helps achieve:
  - Modularity
  - Maintainability
  - Seamless development
  - Real world terms for development

Object Oriented Goals

JOHN BRYCE
Leading in IT Education
a matrix company

Reuse

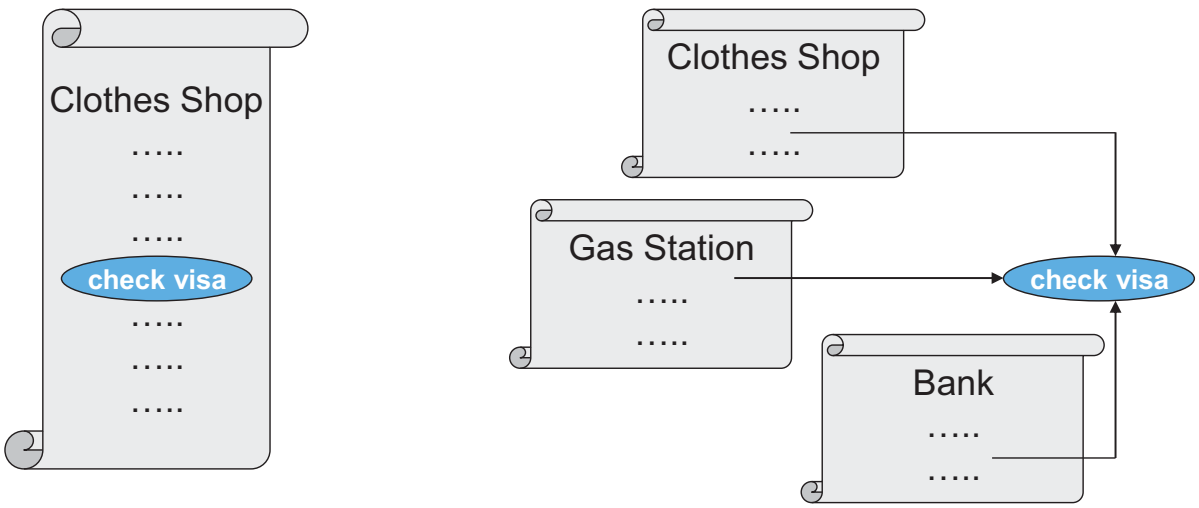
- Code is separated into structured units
- Relations between different unit are well defined
- Each unit can be used in different application flows
- Unit can be adapted & extended by other units
- Analysis and design principles can be also reused

OOP Goals

JOHN BRYCE
Leading in IT Education
a matrix company

Reuse

- When a code is embedded in a particular program it can't be reused for other purposes
- Once it is represented as a stand alone unit it can be reused



Object Oriented Goals

JOHN BRYCE
Leading in IT Education
a matrix company

Maintainability

- Code units can be maintained independently
- In some cases, changes can be vertical
- Good code can be perpetuated

OOP Goals

JOHN BRYCE
Leading in IT Education
a matrix company

Maintainability

- Components are easily maintained since they are
  - Short
  - Focused
  - Structured
- Component update will effect positively its clients / users
- In cases when there is a components relationship – changes on one will effect its relatives in a well defined and known way

Object Oriented Goals

JOHN BRYCE
Leading in IT Education
a matrix company

Ease of Development

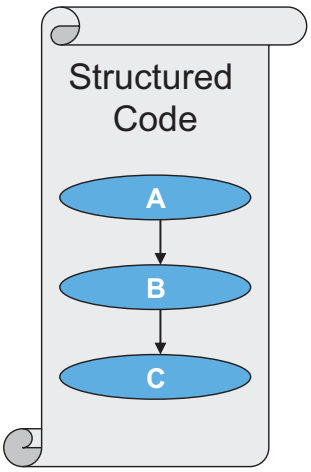
- Code units parallel to real-world units
- Small pieces are combined to implement the hole idea

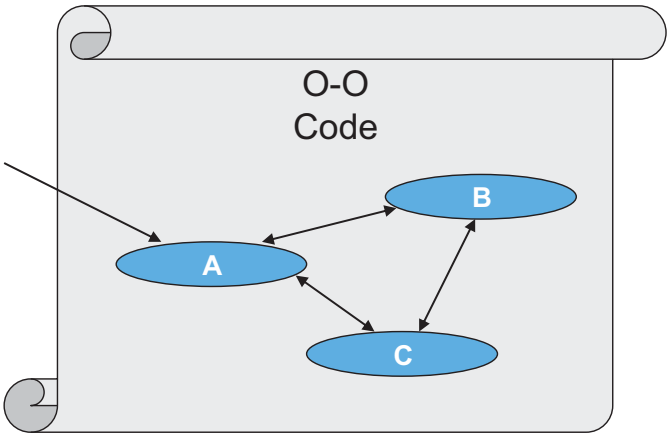
OOP Goals

JOHN BRYCE
Leading in IT Education
a matrix company

Modularity

- Component concept
- Breaking the business logic into small, independent units
- Most units are reusable





OOP Goals

JOHN BRYCE
Leading in IT Education
a matrix company

Seamless development

- Componecan be added and removed
- Other components are allowed to decide when and how to use new components
- Code is more stable
- Code is more nts extensible

Objects

JOHN BRYCE
Leading in IT Education
a matrix company

- Everything can be an object
- A program is a bunch of interacting objects
- Every object has its own memory
- Every object has a state
- Every object has a type

Objects

JOHN BRYCE
Leading in IT Education
a matrix company

Object is a complex variable

- Primitive variable has:
  - One value
  - No operations

```
int x=8;
x = x + 1;
```

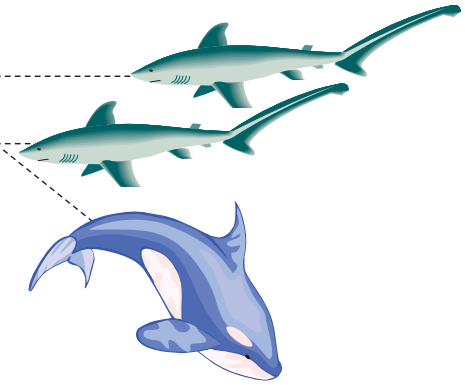
- Objects has:
  - Complex state (out of more than one value)
  - Operations

```
Fish f=…;
f.swim();
```

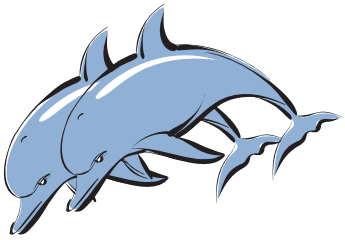

Objects

JOHN BRYCE
Leading in IT Education
a matrix company

- Different objects might
  1. have the same state
  2. be of the same type



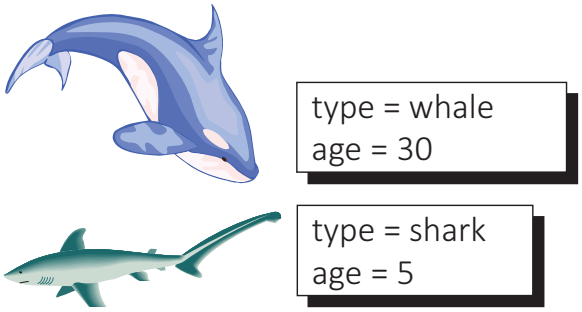
- Different objects can never
  1. have the same memory space
  2. Memory is used for hosting objects state



Objects

JOHN BRYCE
Leading in IT Education
a matrix company

- Object state - the current data held within the object

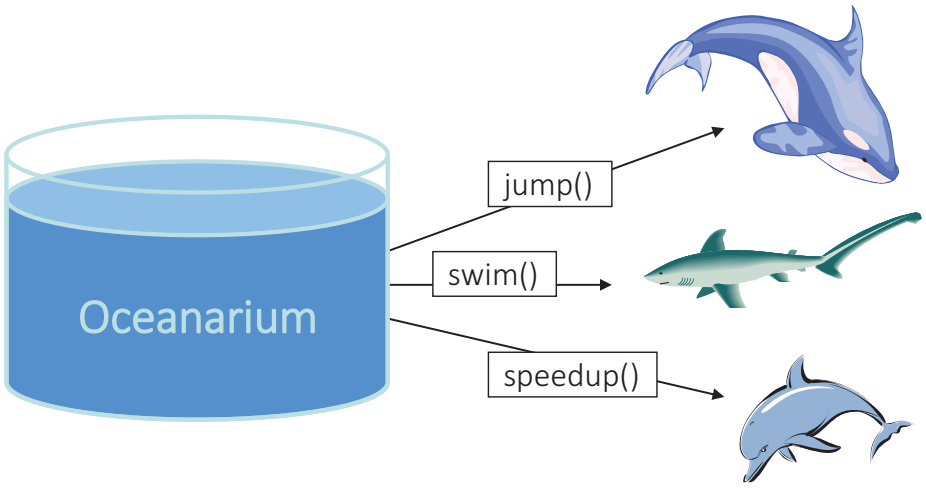


- Object state is held in its attributes
- Object operations may change the object state

Objects

JOHN BRYCE
Leading in IT Education
a matrix company

- Object operations
  - Are the things an object can do
  - Might change the object state
  - Are called via other objects



Objects

JOHN BRYCE
Leading in IT Education
a matrix company

Object type – is specified in a 'class'

- Class defines:
  - Attributes
  - Operations
- Objects of the same type are of the same class and therefore:
  - Have the same infrastructure
  - May vary with their attributes values
- Class is a blueprint / template of Objects

Objects

JOHN BRYCE
Leading in IT Education
a matrix company

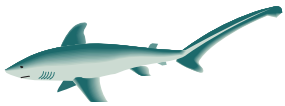
Objects of the same type are different instances of the same class

Fish class

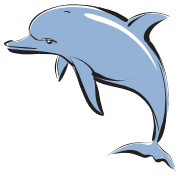
Attributes:
type
age
speed
direction

Operations:
getType
setAge
getAge
speedup
slowdown
turnLeft
turnRight
jump

type = whale
age = 30
speed = 3km/h
direction = down



type = shark
age = 5
speed = 20km/h
direction = straight



type = dolphin
age = 10
speed = 25km/h
direction = up

Objects

JOHN BRYCE
Leading in IT Education
a matrix company

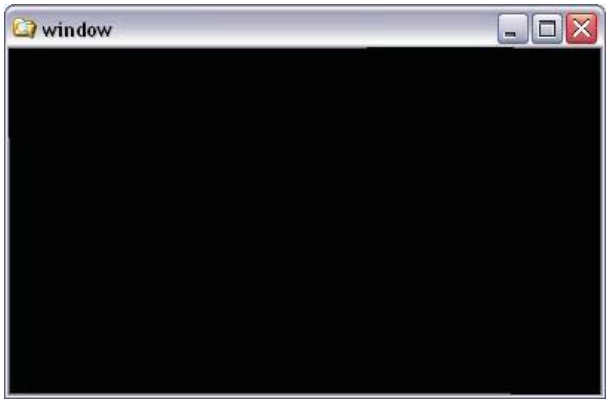
More examples:

Window class

Attributes:
x
y
width
height
bgcolor

Operations:
setLocation
getX
getY
setSize
getWidth
getHeight

x = 100
y = 100
width = 300
height = 150
bgcolor = black



Objects

JOHN BRYCE
Leading in IT Education
a matrix company

More examples:

Account class

Attributes:
balance
dateOfIssue
type

Operations:
withdraw
deposit
getDateOfIssue
getType

balance = 10,000
dateOfIssue = 1.1.2000
type = regular

Objects

JOHN BRYCE
Leading in IT Education
a matrix company

Concrete objects

Beings	Land and Buildings	Equipment	Goods
Person	Store	Car	Directory
Employee	Rental unit	Computer	Book
Customer	Lot	Printer	Boots
Vendor	Parking lot	Telephone	Snowboard
Citizen	Street	Switch	Chair
Tenant	Neighborhood	Cable	Paper
Manager	Farm	Scanner	Coffee

Objects

JOHN BRYCE
Leading in IT Education
a matrix company

Conceptual objects

Organizations	Abstractions	Agreements
Corporation	Strategy	Lease
Division	Plan	Mortgage
Bank	Blueprint	Contract
Sports club	Layout	Covenant
Government department	Proposal	Loan guarantee
Professional association	Map	Warranty

Classes

JOHN BRYCE
Leading in IT Education
a matrix company

Defines a structure and behavior for objects:

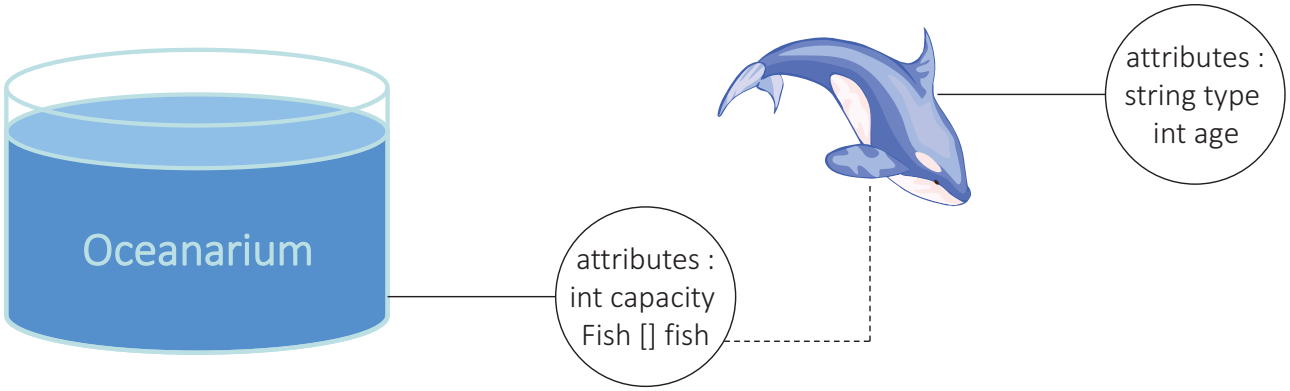
- Attributes – objects state / data
- Operations – object features
- Constructors

Classes

JOHN BRYCE
Leading in IT Education
a matrix company

Attributes

- can be primitives
- can be complex types (classes themselves)

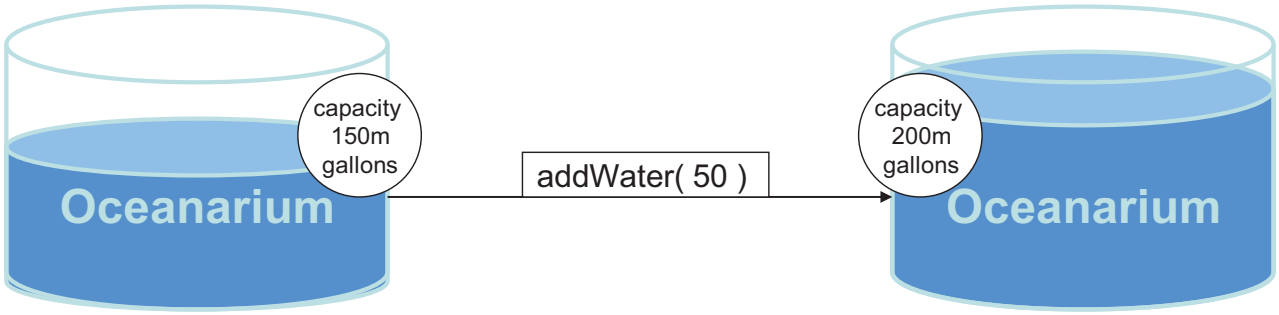


Classes

JOHN BRYCE
Leading in IT Education
a matrix company

Operations

- are the objects features
- may update objects state by assigning values to the objects attributes
- functions – return arguments
- methods – returns void



Classes

JOHN BRYCE
Leading in IT Education
a matrix company

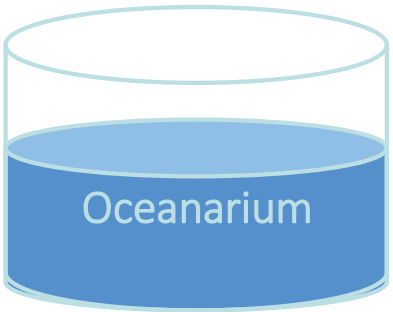
Method overloading

more than one method with the same name in the same class must :

- have same name
- take different arguments
- have the same return type or void

addWater( ) – add 1 gallon to the container

addWater(int gallons) – add the specified amount



Classes

JOHN BRYCE
Leading in IT Education
a matrix company

Constructors

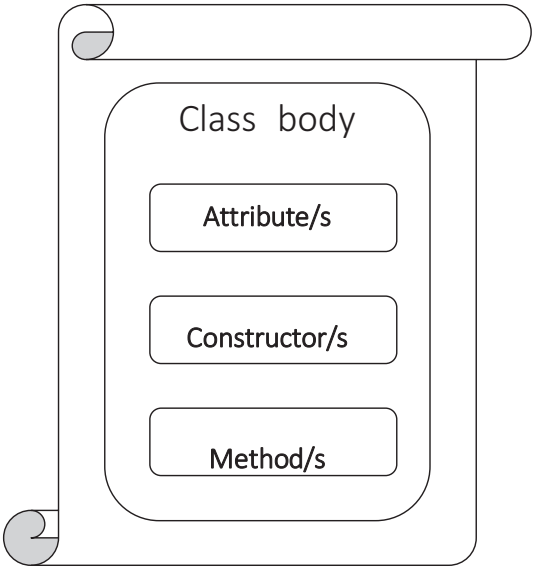
A special method for instantiating objects

- can be called only once during object lifetime
- allows initial state assigning
- can be overloaded (several ways for instantiating)

Classes

JOHN BRYCE
Leading in IT Education
a matrix company

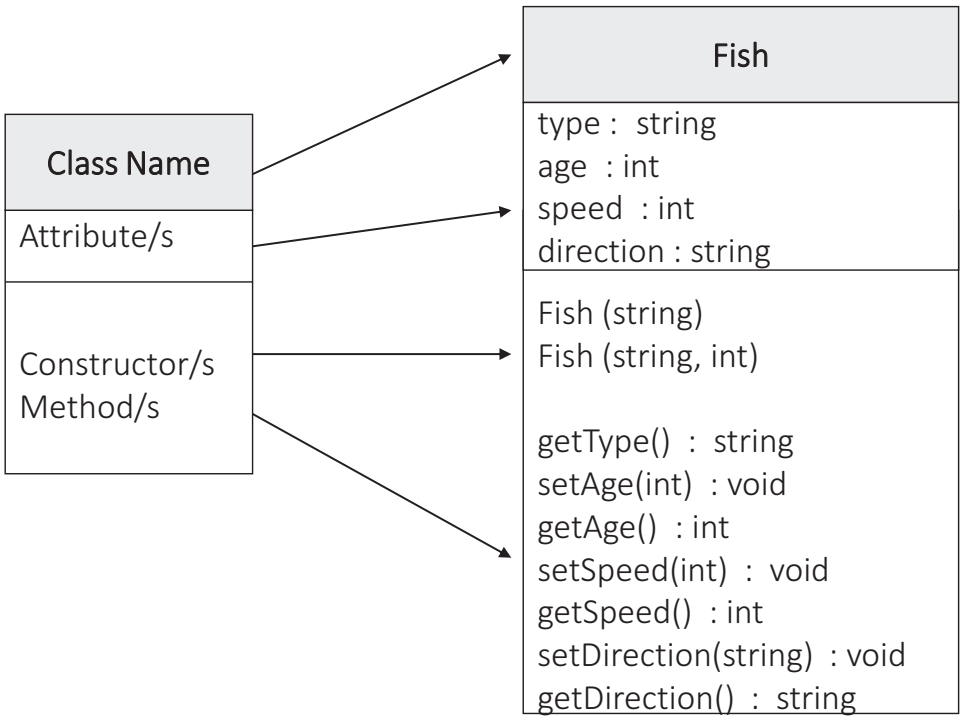
Class infrastructure:



Classes

JOHN BRYCE
Leading in IT Education
a matrix company

UML Class Diagram: Class



Some things to consider

JOHN BRYCE
Leading in IT Education
a matrix company

When designing, planning or coding keep in mind that:

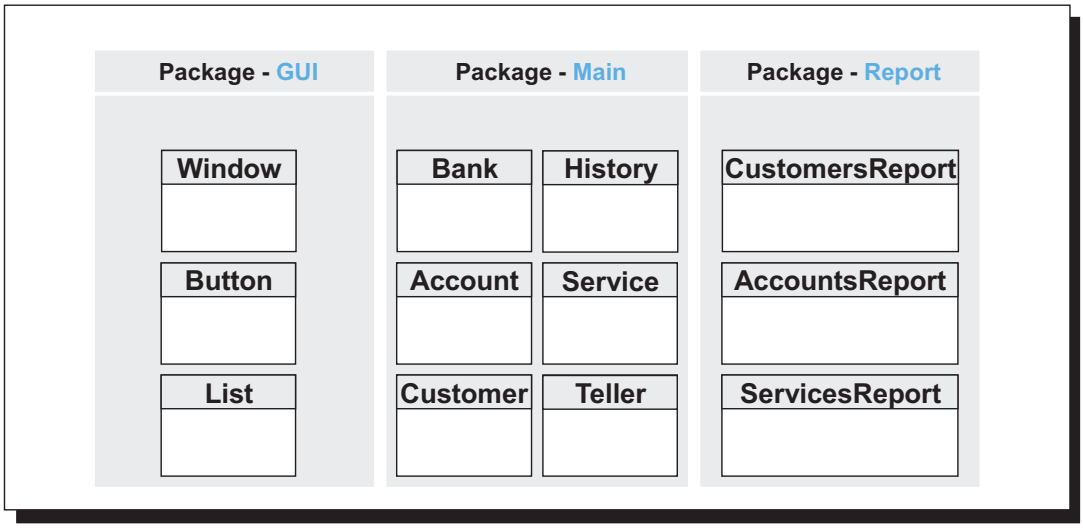
- Classes should describe the functionality required
- High cohesion – objects are targeted and focused
- Clear interface – use logic and convenient operations
- Objects are meant to serve us when providing solutions
- Classes should be reusable for different purposes

Some things to consider

JOHN BRYCE
Leading in IT Education
a matrix company

Packages

- Great amount of classes must be arranged
- Packing classes in directories & subdirectories might help
- Packaging may be supported within the class



Some things to consider

JOHN BRYCE
Leading in IT Education
a matrix company

Method Overloading

- Same operation might use different input parameter
- Operation logic might vary according input parameters

Class PrintToScreen
void print (int value) void print (String value) void print (float value) void print (double value) void print (boolean value) void print (char value) void print (int times, String value)

Rules:

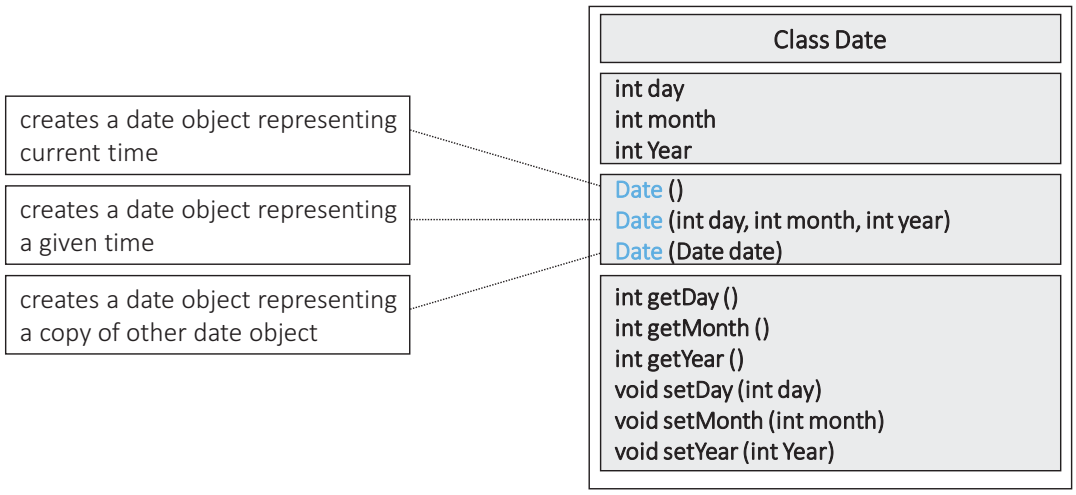
- same name
- different number of input parameters or
- different type of input parameters

Some things to consider

JOHN BRYCE
Leading in IT Education
a matrix company

Constructor overloading

- Exactly like method overloading
- Provides several ways to instantiate objects



Object Oriented Principles

JOHN BRYCE
Leading in IT Education
a matrix company

Encapsulation – the ability to hide class information

Inheritance – the ability to extend classes

Polymorphism – the ability to refer to classes from different hierarchical levels

Encapsulation

JOHN BRYCE
Leading in IT Education
a matrix company

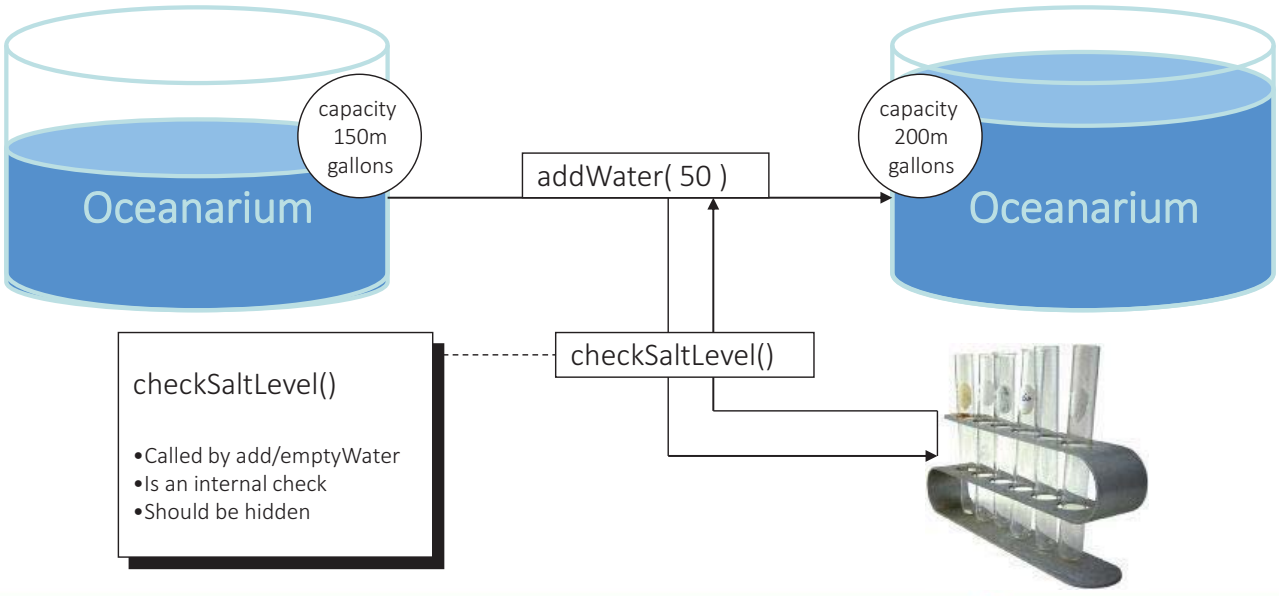
Sometimes data and features should be hidden

- Possible reasons:
  - Information that is not relevant outside the class (used for internal need only)
  - Protecting data
  - Clear interface

Encapsulation

JOHN BRYCE
Leading in IT Education
a matrix company

Information that is not relevant outside the class

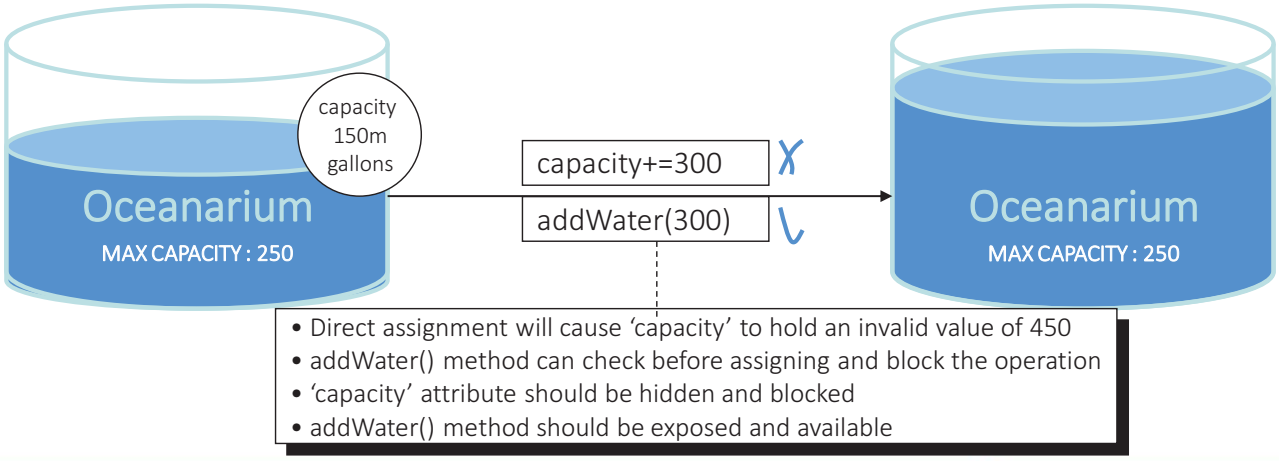


Encapsulation

JOHN BRYCE
Leading in IT Education
a matrix company

Protecting data

- Direct access to attributes might lead to illegal assignments
- Delicate attributes should be hidden
- Accessing attributes values should be via methods
- Access methods can validate the assigned values before updating object state

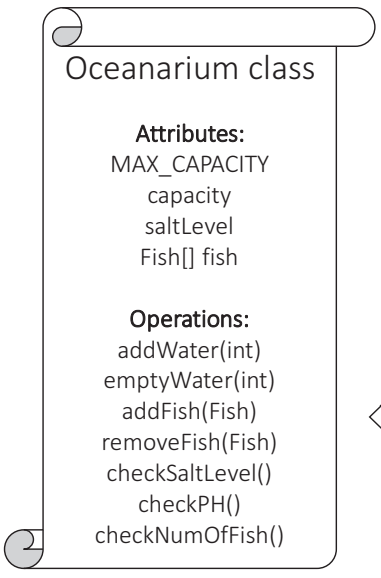


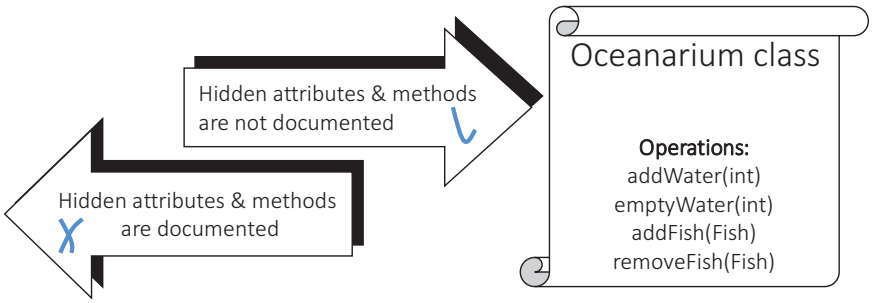
Encapsulation

JOHN BRYCE
Leading in IT Education
a matrix company

Clear interface

- Clients suppose to see only what relevant to them
  - Internal operations and validations will not be exposed
  - Delicate attributes will be also hidden
- Client will get a clear interacting interface with the class
- Documentation should include only what is not hidden





Encapsulation

JOHN BRYCE
Leading in IT Education
a matrix company

Private

- Access modifier for hidden information
- Both attributes & methods can be private
- Private entities are available only within the same class

Public

- Access modifier for exposed information
- Both attributes & methods can be public
- Public entities are available for all

Encapsulation

JOHN BRYCE
Leading in IT Education
a matrix company

UML Class Diagram: Encapsulation

Private : -
Public : +

Oceanarium
- MAX_CAPACITY : int - capacity : int - saltLevel : int - Fish : Fish[]
+ Oceanarium (int) + addWater(int) : void + emptyWater(int) : void + addFish(Fish) : void + removeFish(Fish) : void - checkSaltLevel() : boolean - checkPH() : boolean - checkNumOfFish() : boolean

Inheritance

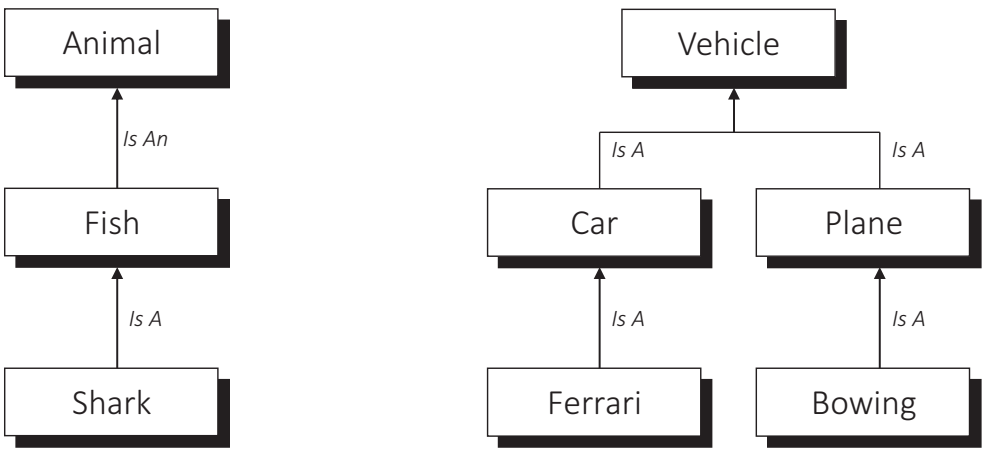
JOHN BRYCE
Leading in IT Education
a matrix company

- Reuse classes implementation and add functionality
- Superclass is the 'parent' of all its inheritors – Subclasses
- Inheritance is used for "Is A" relations between classes
- Why use inheritance?
  - Reuse good code
  - Vertical maintenance – changing Superclass will effect its subclasses
  - Clear interface – each class details only its new features

Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

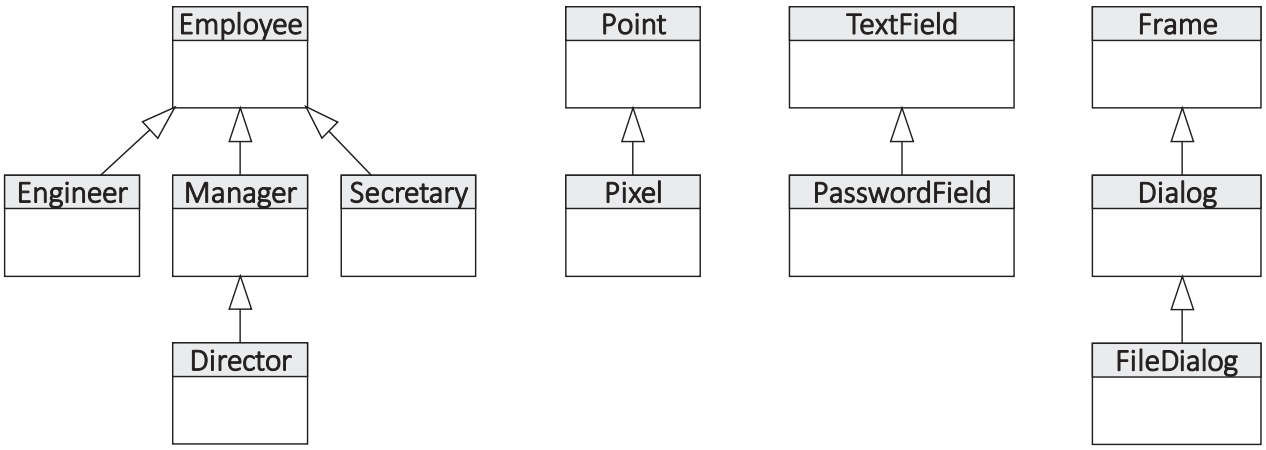
"Is A" relations:



Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

‘ IS A ‘ More Examples



Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

Types of inheritance:

Single inheritance

- each subclass has only one super-class
- it is clear where operations and attributes originated from
- no collisions

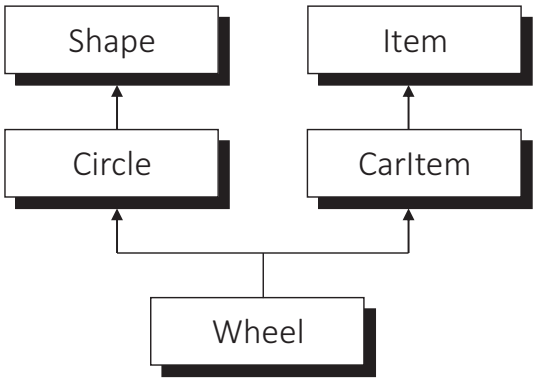
Multiple inheritance

- subclass may inherit several classes
- development is more complex
- collisions might occur

JOHN BRYCE
Leading in IT Education
a matrix company

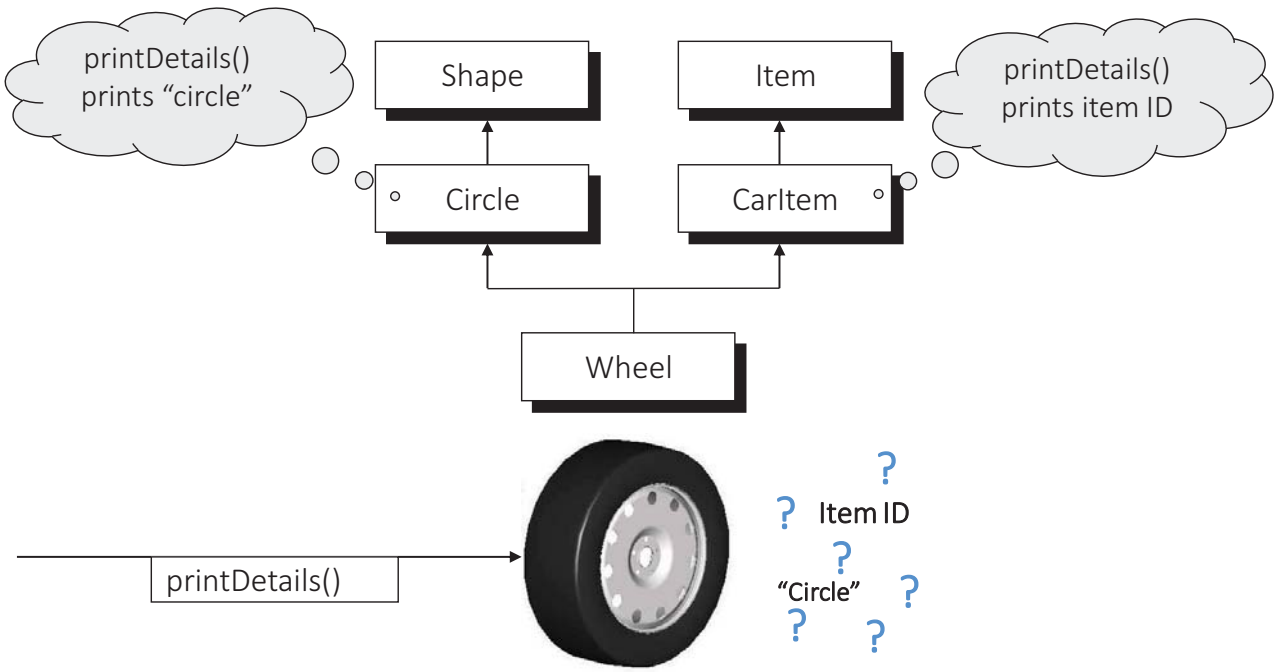
Multiple inheritance - example

Not supported in Java (unlike C++)



JOHN BRYCE
Leading in IT Education
a matrix company

Multiple inheritance maintenance problem:



Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

- Reuse good code
  - Code is inherited / adapted instead of copied
- Clear interface
  - Each class hold only its enhancements
- Vertical maintenance
  - Superclass upgrades are automatically delegated to its subclasses
- infinite extensibility
  - The class hierarchy is endless

Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

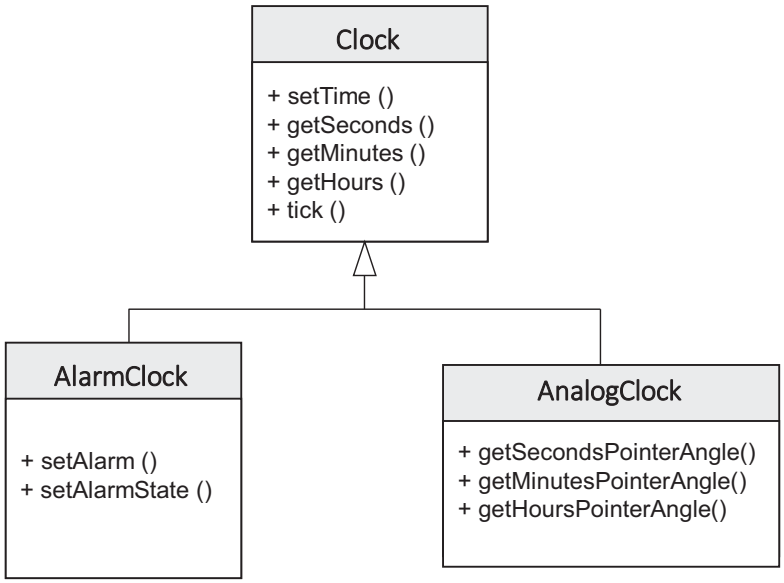
Benefits:

- reusability of successful code
- changes on super-classes are valid for subclasses
- subclasses interface remains clear – specifying new features only
- infinite extensibility

Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

Singe inheritance example:



Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

What is not inherited?

- Constructors
- Private attributes & operations (inherited but not reachable)

JOHN BRYCE
Leading in IT Education
a matrix company

Inheritance and encapsulation

	class	subclass	package	universe
public	○	○	○	○
protected	○	○	○	–
default	○	–	○	–
private	○	–	–	–

JOHN BRYCE
Leading in IT Education
a matrix company

Method overriding

- When rewriting inherited methods in subclasses
- Why should we override super-class methods ?
  - Change the operation behavior
  - Expand the super-class implementation
  - Vetoing super-class activity

JOHN BRYCE
Leading in IT Education
a matrix company

Method overriding

Rules for method overriding

- Method must have the same name
- Method must return the same type
- Method must have the same input parameters

JOHN BRYCE
Leading in IT Education
a matrix company

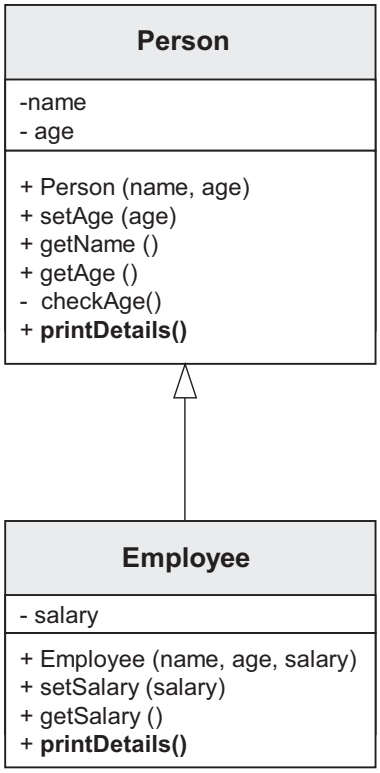
Method overriding

```
public class Person {
   ….
   public void printDetails(){
          print (name) ;
          print (age);
   }
   …
}
```

```
public class Employee extends Person {
   ….
   public void printDetails(){
          super.printDetails();
          print (salary);
   }
   …
}
```

or

```
public class Employee extends Person {
   ….
   public void printDetails(){
          print (getName()) ;
          print (getAge());
          print (salary);
   }
   …
}
```



Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

Problem in multiple inheritance:

- When two super-classes have the same method signature
- Sub-class call to super.method() is ambiguous

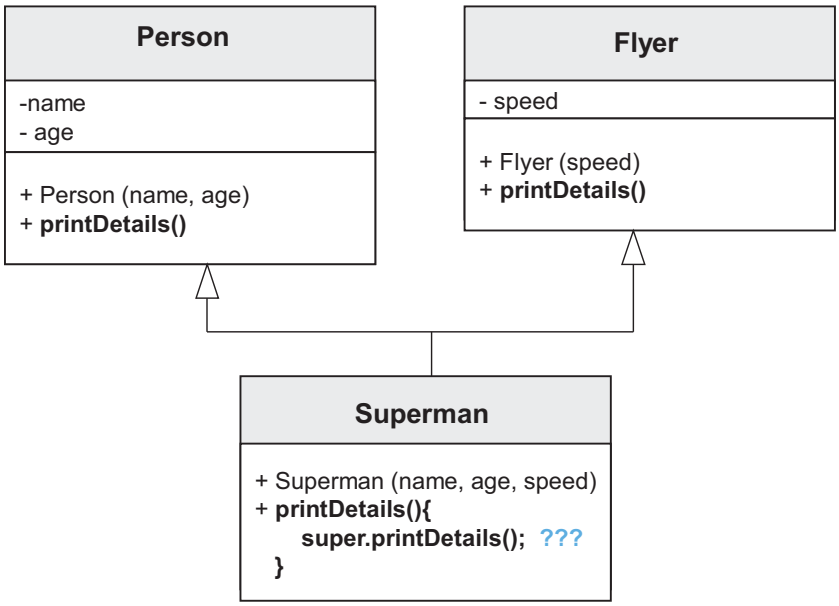
Solution:

- Specify which super-class is used to call super.method()
- Makes development & debugging more complex & problematic

Inheritance

JOHN BRYCE
Leading in IT Education
a matrix company

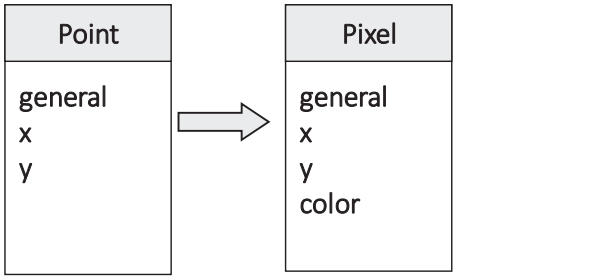
Problem in multiple inheritance – example:



Polymorphism

JOHN BRYCE
Leading in IT Education
a matrix company

An instance can be referenced from a higher hierarchy point:




```
Pixel p= new Pixel(1,5,"Red");
```

```
Point p= new Pixel(1,5,"Red");
```


Polymorphism

JOHN BRYCE
Leading in IT Education
a matrix company

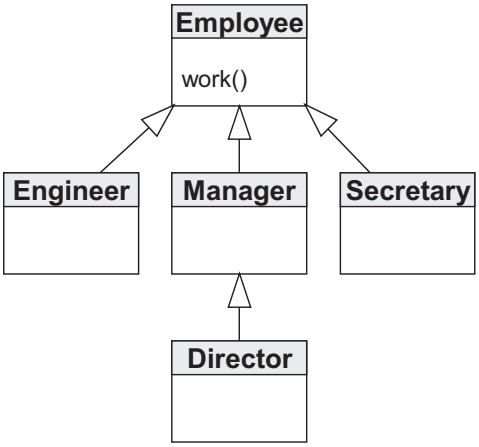
Why do we need polymorphism?

- Treat different hierarchy nodes in the same way
- Define operations for existing and future classes

Polymorphism

JOHN BRYCE
Leading in IT Education
a matrix company

Treat different hierarchy nodes in the same way

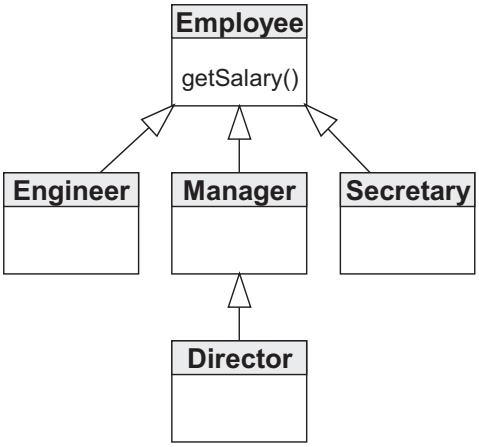



```
Employee [ ] emp = …

emp [0] = new Employee();
emp [1] = new Engineer();
emp [2] = new Manager();
emp [3] = new Secretary();
emp [4] = new Director();

for (int i=0;i<emp.length;i=i+1){
      emp.work();
}
```


Polymorphism

JOHN BRYCE
Leading in IT Education
a matrix company

Define operations for existing and future classes



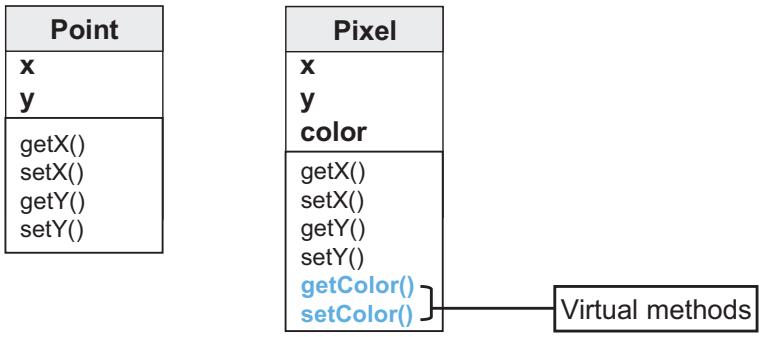

```
int calculateYearlySalary ( Employee emp) {
     total=emp.getSalary() * 12 ;
     return total;
}

..
..

Manager manager=new Manager();
int total = calculateYearlySalary (manager);
```

JOHN BRYCE
Leading in IT Education
a matrix company

Virtual Methods

- The Pixel variable p can invoke all the methods of Pixel:

```
Pixel p= new Pixel(1,5,"Red");
```

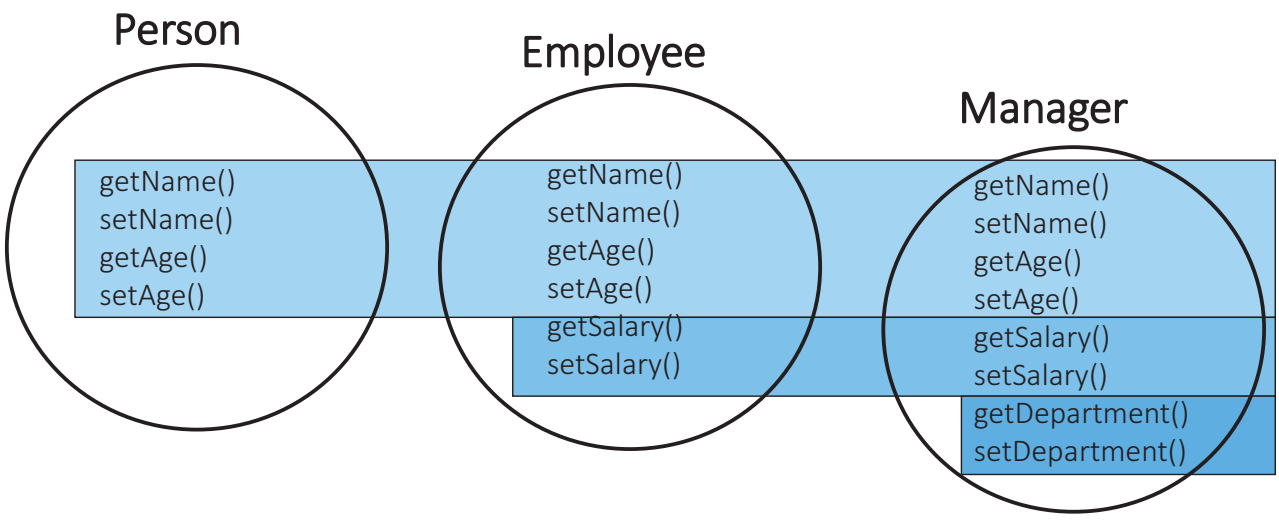


- The Point variable p can invoke only Point methods

```
Point p= new Pixel(1,5,"Red");
```


JOHN BRYCE
Leading in IT Education
a matrix company

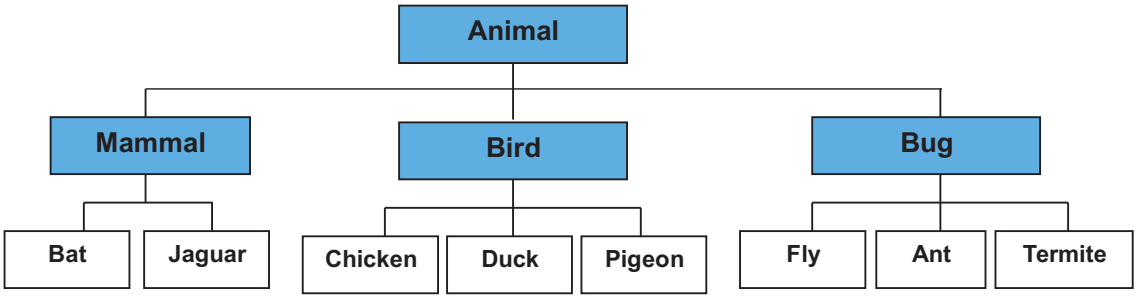
Virtual Methods



Abstract

JOHN BRYCE
Leading in IT Education
a matrix company

- Sometimes we don't know how to implement a behavior or an operation
- It is often useful to think in a real-world terms
- Objects cannot be created from abstract classes
- Concrete subclasses must override abstract methods to become non-abstract



Abstract

JOHN BRYCE
Leading in IT Education
a matrix company

Animal

- There is no such a thing Animal
- Animal specifies the basic behavior of living creatures
- There are some animal operations that we know how to implement:

```
getName()
getAge()
```

- And there are other animal operations that might be implemented differently in different animals:

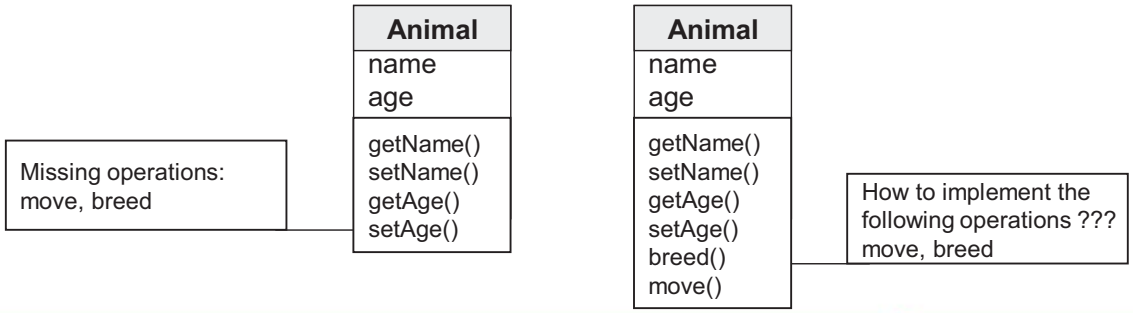
```
move()
breed()
```

Abstract

JOHN BRYCE
Leading in IT Education
a matrix company

The problem:

- If we will define only the methods that we know how to implement than subclasses might ignore the need to specify the other methods
- We want to force a behavior on subclasses but we don't know how to implement it



Abstract

JOHN BRYCE
Leading in IT Education
a matrix company

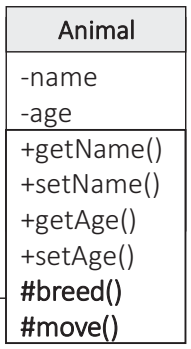
The solution:

- Define the problematic method as abstract methods
- Abstract methods:
  - Has no body (no implementation)
  - Must be overridden by concrete subclasses
  - Turns the class to be also abstract

Abstract

JOHN BRYCE
Leading in IT Education
a matrix company

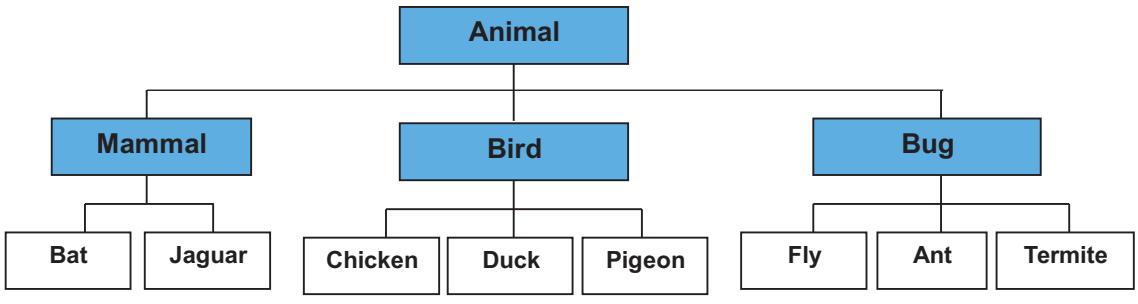
```
public abstract class Animal {

   ...
   ...
   public abstract void breed () ;

   public abstract void move () ;
   ...
}
```



Abstract

JOHN BRYCE
Leading in IT Education
a matrix company

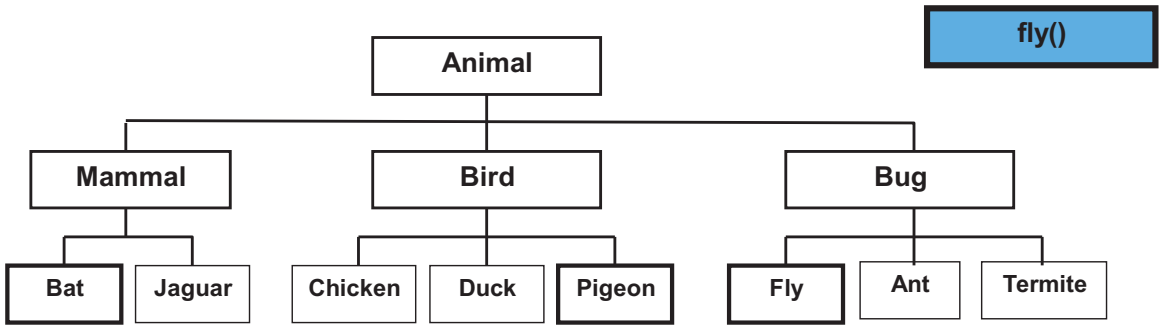
- Up the hierarchy we will find more abstract classes
- Down the hierarchy we will find more concrete classes that implements the inherited abstract behavior



Interfaces

JOHN BRYCE
Leading in IT Education
a matrix company

The problem:



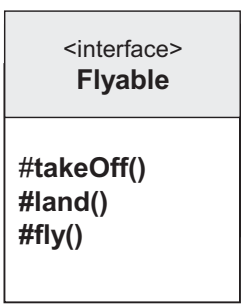
Where should we place the abstract method *fly()*?

- Animal – most of the animal don't support this operation
- Bird – not all birds are capable of flying
- There are some mammals & insects that can fly

Interfaces

JOHN BRYCE
Leading in IT Education
a matrix company

- Contains only abstract methods
- May define data members
- Class that implements an interface must
  - Override all methods or
  - Declared as abstract



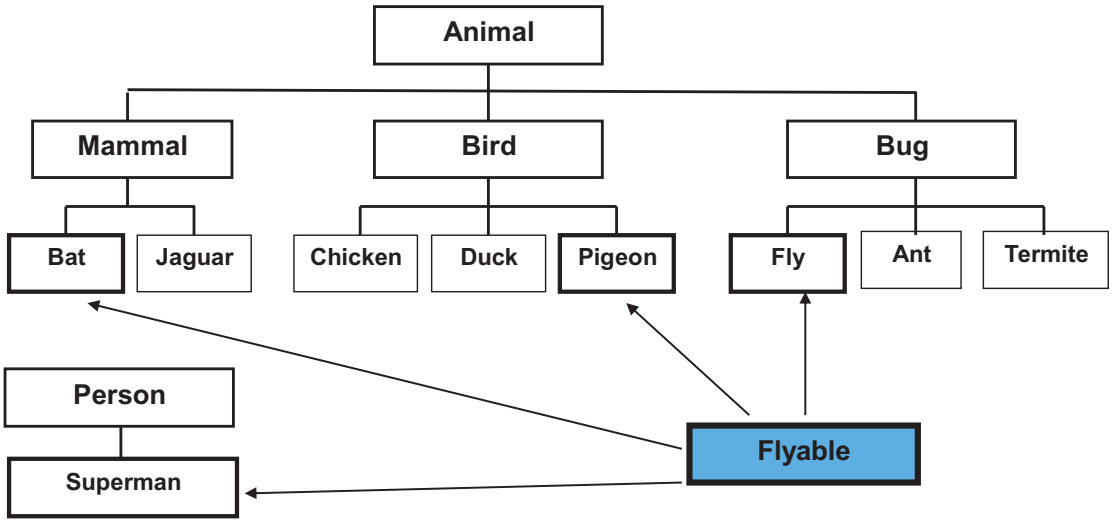
Interfaces

JOHN BRYCE
Leading in IT Education
a matrix company

- Are polymorphic point of view
- Are external to the class inheritance family
- Enables classes from different families have some common operations
- A class may implement more than one interface

Interfaces

JOHN BRYCE
Leading in IT Education
a matrix company



Interfaces

JOHN BRYCE
Leading in IT Education
a matrix company

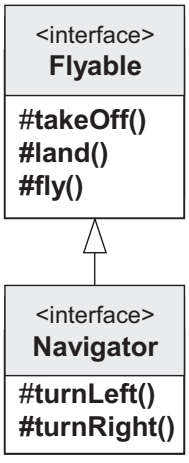
Interfaces as a polymorphic point of view

```
Flyable [] flyers=new Flyable[4];

Flyers[0]=new Superman();
Flyers[1]=new Bat();
Flyers[2]=new Fly();
Flyers[3]=new Pigeon();


for (int i=0;i<4;i++){
    flyer[i].takeOff();
    flyer[i].fly();
}
```


Interfaces

JOHN BRYCE
Leading in IT Education
a matrix company

- Interfaces can inherit other interfaces
- Interfaces can inherit more than one interface
- The class implements sub-interface must override all the inherited methods



Interfaces

JOHN BRYCE
Leading in IT Education
a matrix company

- Always prefer 'programming to an abstract classes or interfaces'
- Interfaces can become higher level protocol between to parts of the application or system
- Interfaces can be used to define standards (APIs) & models

Adapters

JOHN BRYCE
Leading in IT Education
a matrix company

Sometimes a class implements a multi-method interface

Problem:

- the class need to implement only several out of many methods declared in the interface

Solution:

- an adapter class will implement the interface and override all it methods to do nothing (empty methods – with no body)
- The concrete class will extend the adapter and override the several wanted method leaving the others empty

Adapters

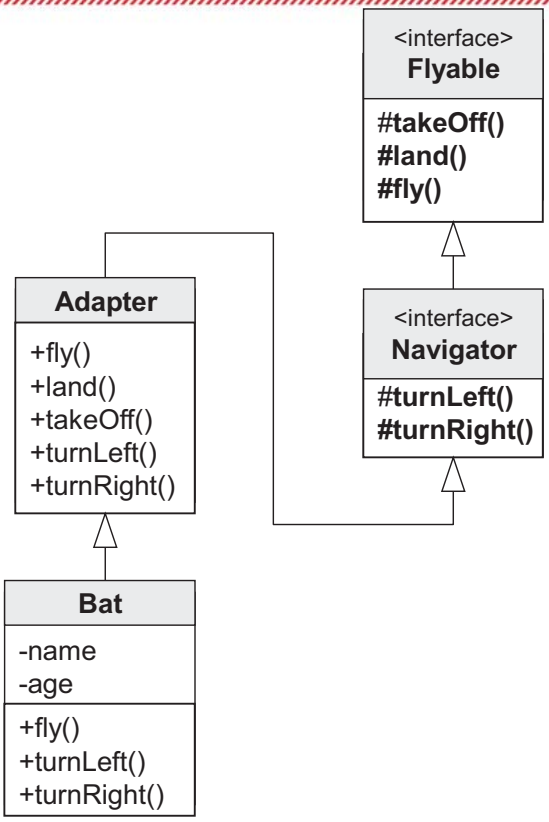
JOHN BRYCE
Leading in IT Education
a matrix company

Bat supports:

- fly
- turn

Bat doesn't support

- land
- take off



Final

JOHN BRYCE
Leading in IT Education
a matrix company

- Classes, methods and variables can be defined as final
- Final class – cannot be inherited
  - Usually required for system classes
  - Some basic behaviors that mustn't be extended or changed
- Final method cannot be overridden
  - Forces sub-classes to use a specific implementation
- Final variable can have only one assignment [constant]
  - Defines a constant values
  - Local variables can also be defined as final

Static

JOHN BRYCE
Leading in IT Education
a matrix company

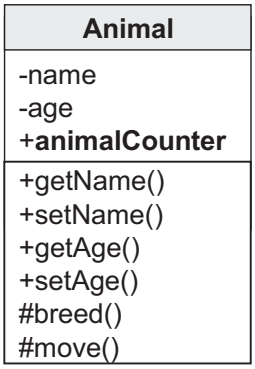
- Static defines entities at the class level
- Variables and operation can be static
- Static entities can live without object allocations
- Static mustn't depend on any non-static content

Static

JOHN BRYCE
Leading in IT Education
a matrix company

Static variables

- Are class members
- Live beyond objects lifetime
- Loaded into memory and initialized only once – when first used
- Used and referenced without any object allocation



Static

JOHN BRYCE
Leading in IT Education
a matrix company

Static variables

```
public abstract class Animal {
      public static int animalCounter;
      …
      public Animal(){
            …
            animalCounter++;
      }
      …
}
```

```
print ( Animal.animalCounter );   0

Animal a=new Bat();
Animal b=new Jaguar();
Animal c=new Cat();

print ( Animal.animalCounter );   3
```


Static

JOHN BRYCE
Leading in IT Education
a matrix company

Static methods

- Are called directly from the class (no objects are needed)
- Can work
  - independently
  - with other static content (methods or members)

Examples for static methods:

- A method that takes a parameter and converts it
- A method that takes two numbers and returns the sum
- A method that returns the animalCounter value

Static

JOHN BRYCE
Leading in IT Education
a matrix company

Static methods

Animal
-name -age **-animalCounter**
+getName() +setName() +getAge() +setAge() #breed() #move() **+getAnimalCountert()**

```
public abstract class Animal {
    private static int animalCounter;
    …
    public Animal(){
        …
        animalCounter++;
    }
    …
    public static int getAnimalCounter(){
        return animalCounter;
    }
}
```

```
print ( Animal. getAnimalCounter() );   0

Animal a=new Bat();
Animal b=new Jaguar();
Animal c=new Cat();

print ( Animal. getAnimalCounter() );   3
```


Final - Constant

JOHN BRYCE
Leading in IT Education
a matrix company

Final / Constant

- For variables – means that only one assignment can be done
- For classes – Final classes cannot be inherited
- For methods – Final methods cannot be overridden

Exercise

Design a Bank system

- 3 types of clients - regular, gold & platinum
  - each client holds a collection of accounts
  - Can add/remove account
  - Account reports
    - Total accounts
    - Total Money
- Each account supports
  - Current amount
  - Deposit operation
  - Withdraw operation
  - Status (allowed, warned, blocked)
- The bank supports
  - Adding / removing clients
  - Reports:
    - Client list report
    - Total accounts report
    - Total money report
    - Total activity (clients and account operation summery)