UML Class Diagram

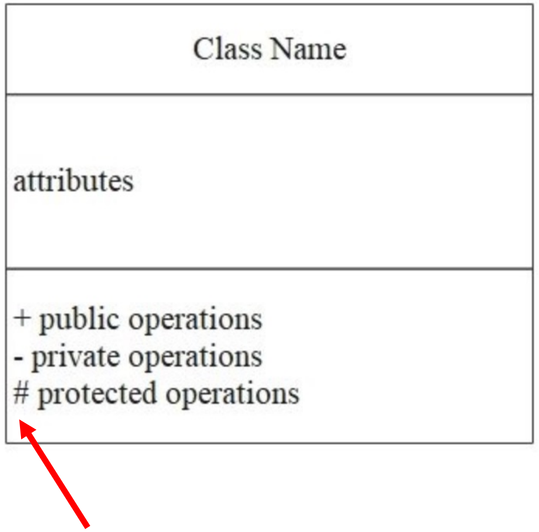
The usage of Class Diagram

The class diagram are typically used:

- to map out what a system would conceptually look like in static form.
- to plan the development of the system. They can be easily amended.
- to outline the relationships between classes
- to allow an analysis of the system as a whole

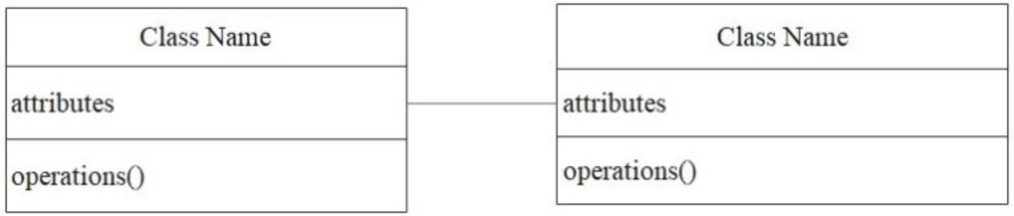
Symbols used when constructing the diagram

Class

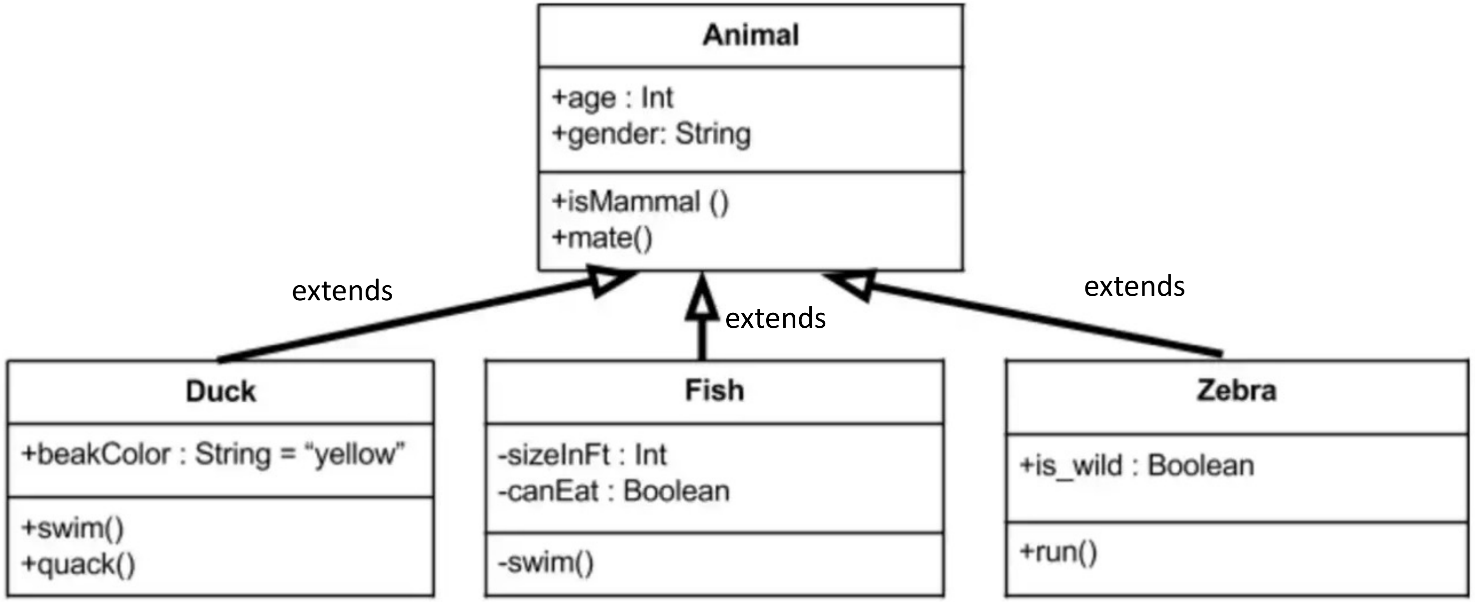


Visibility symbols are used to determine the accessibility of the information contained in classes.
*The “**+**” represents public operations, vice versa, “**-**” represents private operations. Plus, “**#**” is for the protected operations.*

Bidirectional association: represented by a straight line connecting two classes. It simply demonstrates that the classes are aware of their relationship with each other.

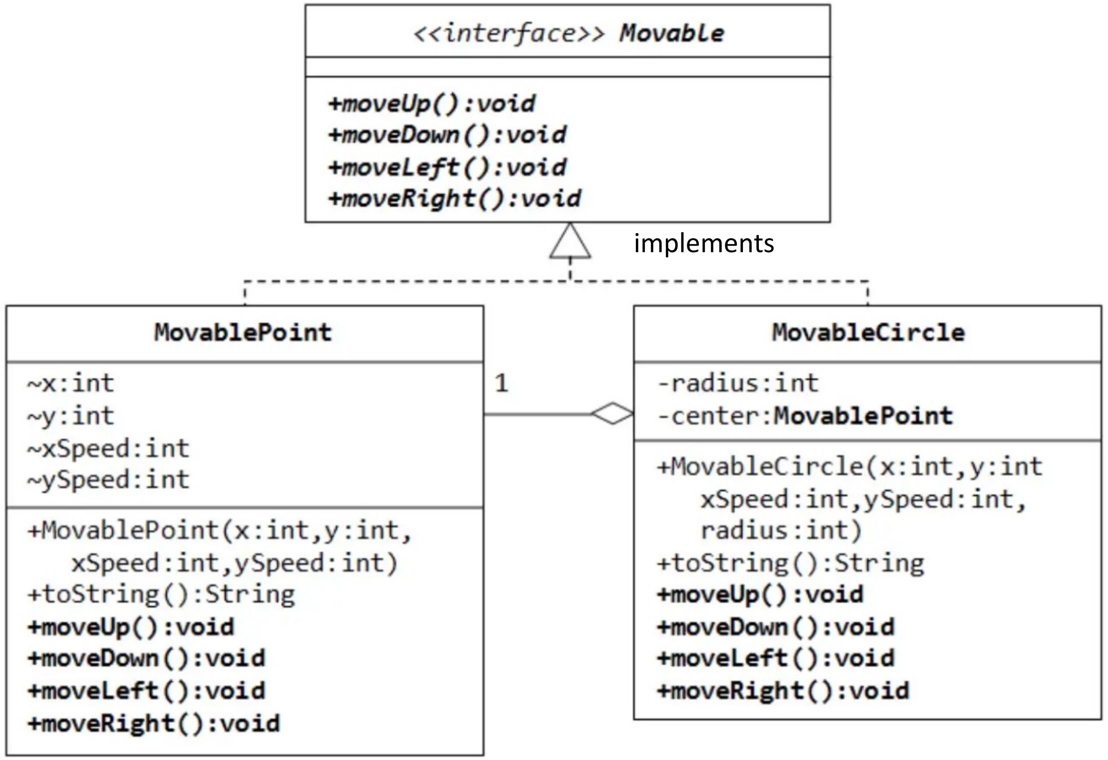


Inheritance Relationships between classes



Indicate a “child-parent” relationship between classes. The child class is a specialized, sub-class of the parent.

Interface



*<<interface>> **Movable***

+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void

implements

MovablePoint

~x:int
~y:int
~xSpeed:int
~ySpeed:int

+MovablePoint(x:int,y:int,
xSpeed:int,ySpeed:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void

1

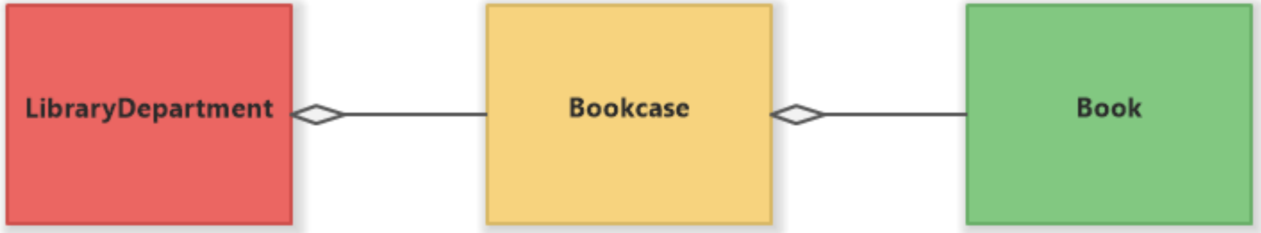
MovableCircle

-radius:int
-center:**MovablePoint**

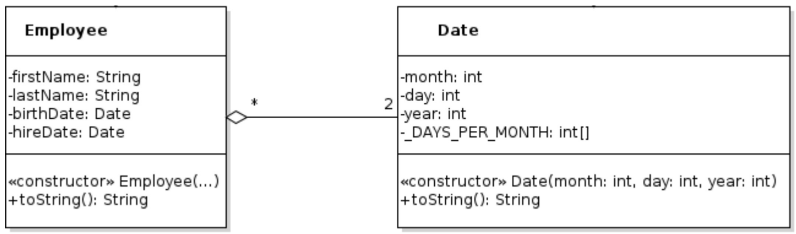
+MovableCircle(x:int,y:int
xSpeed:int,ySpeed:int,
radius:int)
+toString():String
+moveUp():void
+moveDown():void
+moveLeft():void
+moveRight():void

One class implements the behaviour specified by another class.

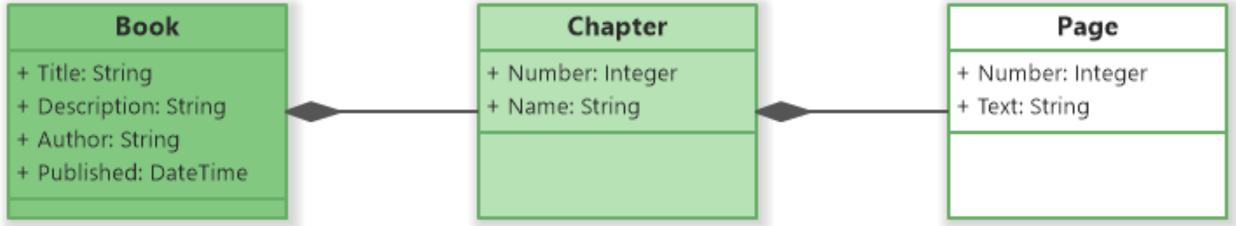
Aggregation



This represents a unilateral relationship between classes. One class is part of, or subordinate to, another. In this instance, the child and parent classes can exist independently.

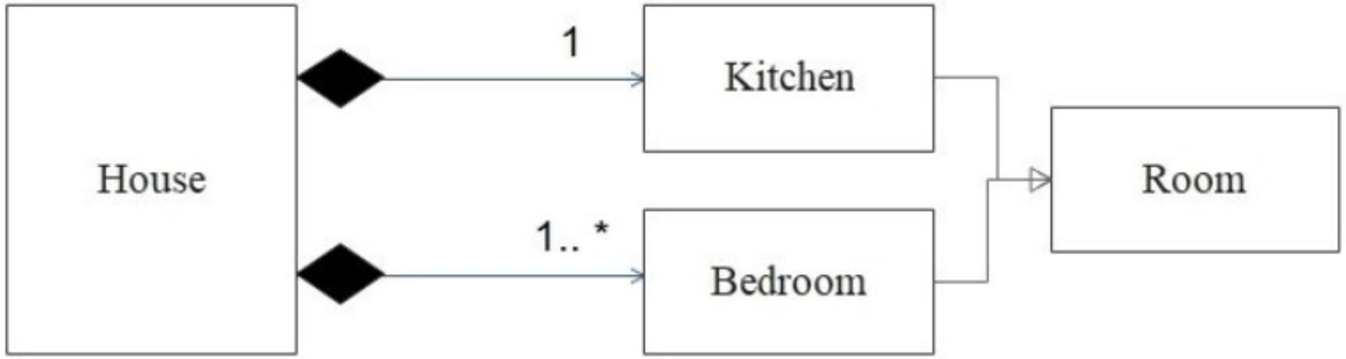


Composition

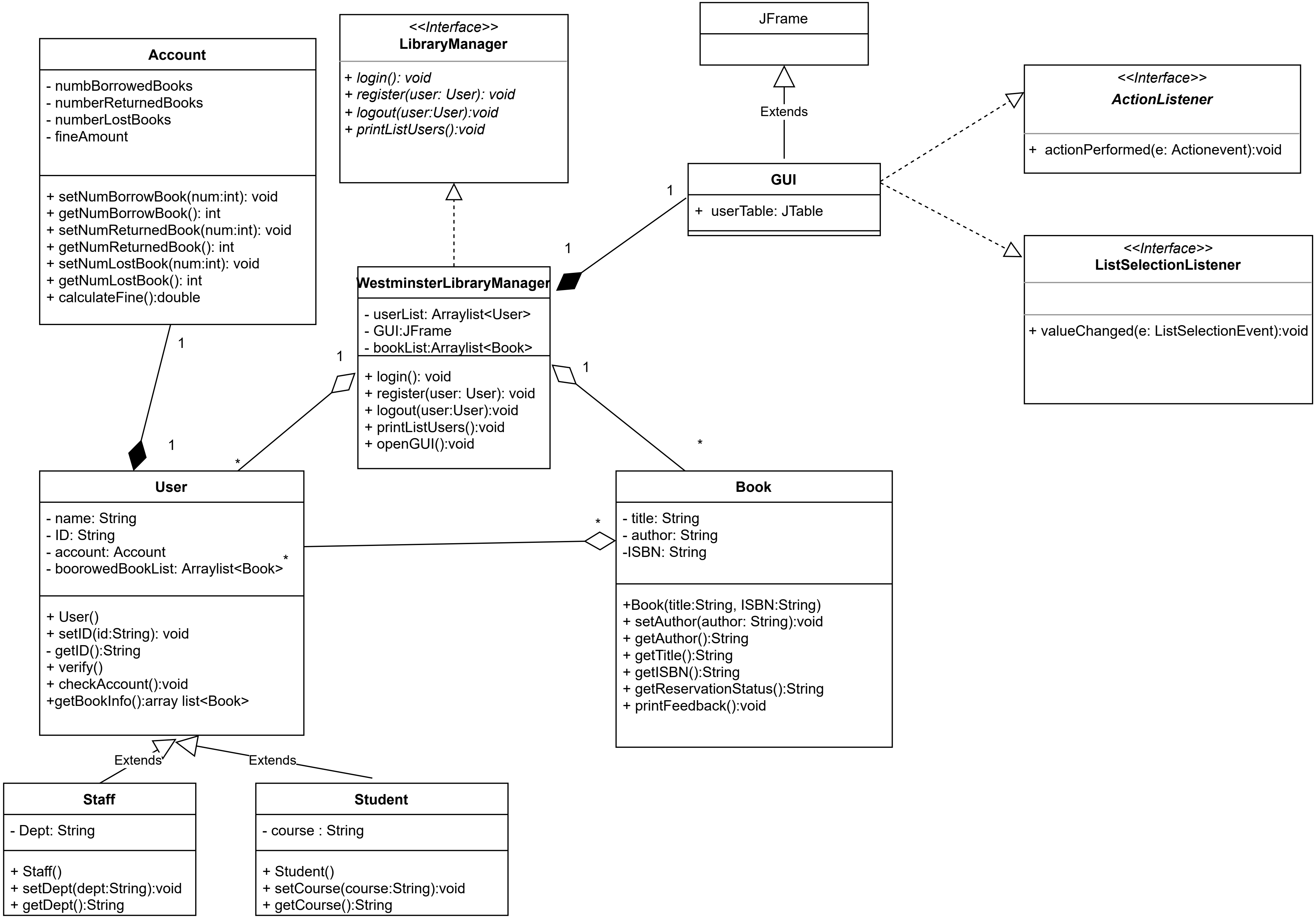


It is a form of aggregation where one class is dependent on another. One class is a part of the other. In this instance, the child classes and parent classes cannot exist independently.

Multiplicity



Multiplicity is used to determine how many times an attribute occurs.
In this example, this house has exactly one kitchen and at least one bedroom.


Account
- numbBorrowedBooks
- numberReturnedBooks
- numberLostBooks
- fineAmount
+ setNumBorrowBook(num:int): void
+ getNumBorrowBook(): int
+ setNumReturnedBook(num:int): void
+ getNumReturnedBook(): int
+ setNumLostBook(num:int): void
+ getNumLostBook(): int
+ calculateFine():double
<<Interface>>
LibraryManager
+ login(): void
+ register(user: User): void
+ logout(user:User):void
+ printListUsers():void
JFrame
Extends
GUI
+ userTable: JTable
<<Interface>>
ActionListener
+ actionPerformed(e: Actionevent):void
<<Interface>>
ListSelectionListener
+ valueChanged(e: ListSelectionEvent):void
WestminsterLibraryManager
- userList: Arraylist<User>
- GUI:JFrame
- bookList:Arraylist<Book>
+ login(): void
+ register(user: User): void
+ logout(user:User):void
+ printListUsers():void
+ openGUI():void
1
1
1
1
1
1
1
*
*
*
*
User
- name: String
- ID: String
- account: Account
- boorowedBookList: Arraylist<Book>
+ User()
+ setID(id:String): void
- getID():String
+ verify()
+ checkAccount():void
+getBookInfo():array list<Book>
Book
- title: String
- author: String
-ISBN: String
+Book(title:String, ISBN:String)
+ setAuthor(author: String):void
+ getAuthor():String
+ getTitle():String
+ getISBN():String
+ getReservationStatus():String
+ printFeedback():void
Extends
Extends
Staff
- Dept: String
+ Staff()
+ setDept(dept:String):void
+ getDept():String
Student
- course : String
+ Student()
+ setCourse(course:String):void
+ getCourse():String