Bahria University Karachi Campus

CSC- 210
Object Oriented Programming

Lecturer
Sameena Javaid

https://sites.google.com/site/sameenajavaidcs

LECTURE 06

CLASS RELATIONSHIPS

<u>OUTLINE</u>

- Array of Objects
- Class Relationships
- Association
- Aggregation
- Composition

ARRAY OF OBJECTS

ARRAY OF OBJECTS

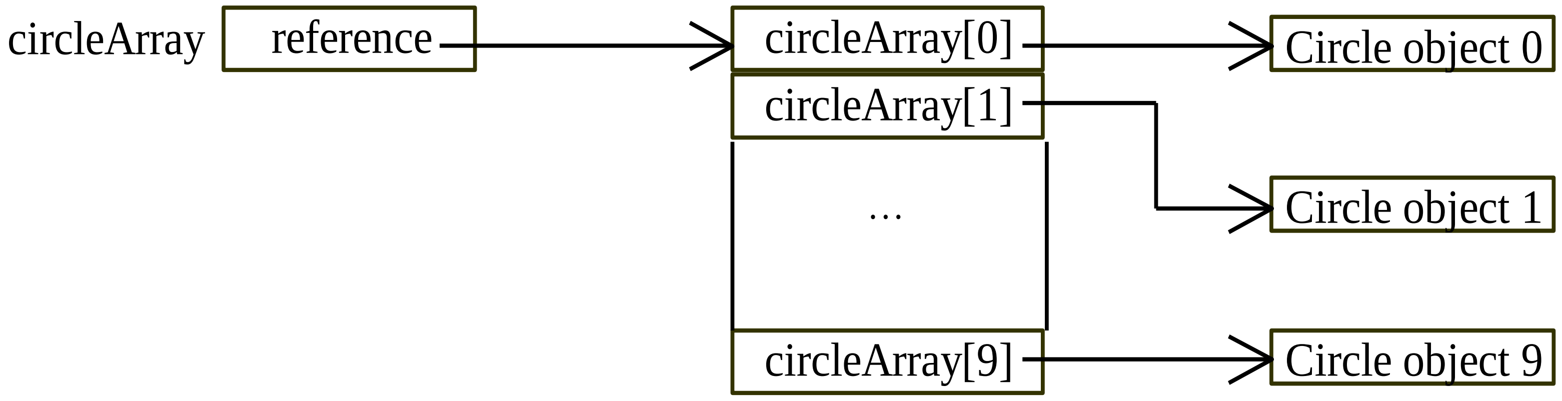
- An array of primitive data is a powerful tool, but an array of objects is even more powerful.
- An array of objects is actually an array of *reference variables*.
- The use of an array of objects allows us to model the application more cleanly and logically.

```
Circle[] circleArray;
circleArray = new Circle[10];
```

- So invoking **circleArray[1].findArea()**involves two levels of referencing as shown in the next figure.
  - circleArray references to the entire array.
  - circleArray[1] references to a Circle object.

ARRAY OF OBJECTS

```
Circle[] circleArray = new Circle[10];
```



ARRAY OF OBJECTS: EXAMPLE

- Assumed that we have a **`Book`** class as shown in a class diagram below:

```
Book
--------------------------------
-title:String
-author:String
-yearPublished:int
--------------------------------
+Book()
+Book(String,String,int)
+setTitle(String):void
+setAuthor(String):void
+setYearPublished(int):void
+getTitle():String
+getAuthor():String
+getYearPublished():int
```

ARRAY OF OBJECTS: EXAMPLE

- We can create an object from **Book** class to store information of one book by using the following statements:

Book b = new Book(title, author, yearPublsihed);

.....call other methods.........

- What if we have to get the information of 5 books?
- What if we have to get the information of 10 books?
- What if we have to get the information of 100 books?
- **How to store information of more than 100 book?**

ARRAY OF OBJECTS: EXAMPLE

- Declare and create an array of object is just like we declare and create an array of primitive data type

  - **Syantax:**
    ClassName [] ArrayName = new ClassName[size];

  - **Example:**
    Book[] b = new Book[20]; 1st statement
    Movie[] movie = new Movie[8]; 2nd statement
    Candy[] candy = new Candy[100]; 3rd statement

Array of objects: Example: Book

```
public class Book {

  private String title;
  private String author;
  private int yearPublished;

  public Book(){} //default constructor
   public Book(String t, String a, int y) {
     title = t; author = a; yearPublished = y;
   }
  //mutator method
  public void setBook(String title){this.title = title;)}
   public void setAuthor(String author){this.author = author;)}
   public void setYearPublished(int yearPublished){
  this.yearPublished = yearPublished;)}

  //accessor method
  public String getTitle(){return title;}
  public String getAuthor(){ return author;}
  public int getYearPublished(){return yearPublished; }
}
```

Array of objects: Example: Book

```
   public class BookTest {
   public static void main(String[] args){
      String title, author; int year;
       Scanner input = new Scanner(System.in);

A     Book book[];

B     book = new Book[20];
      for(int i=0;i<book.length;i++){
            //get all the 3 values of attributes
            title= input.nextLine();
            author= input.nextLine();
            year=  input.nextInt();
C           book[i]=new Book(title, author, year);
      }
      //display
      for(int j=0;j<book.length;j++){
            System.out.println(book[j].getTitle());
            System.out.println(book[j].getAuthor());
            System.out.println(book[j].getYearPublished());
      }
   }
```

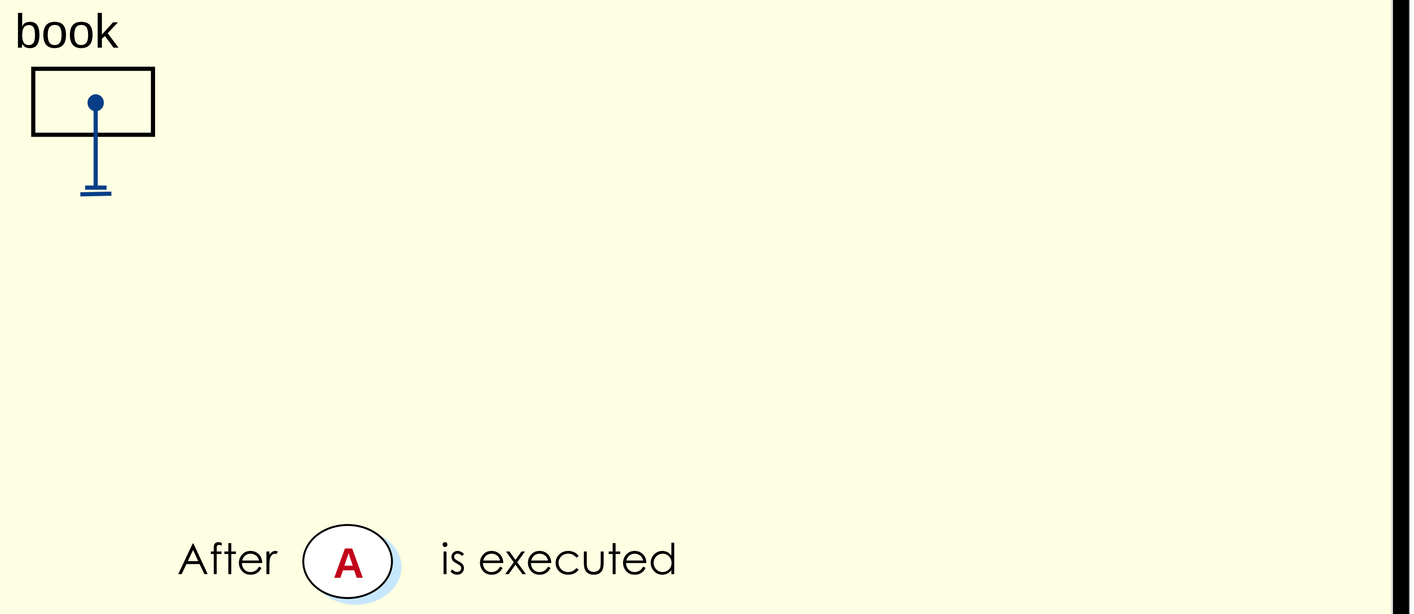
Creating an Object Array - 1

Code A

```
Book book [];
book = new Book[20];
book[0] = new Book(title, auth, year);
```

Only the name book is declared, no array is allocated yet.

State of Memory



Creating an Object Array - 2

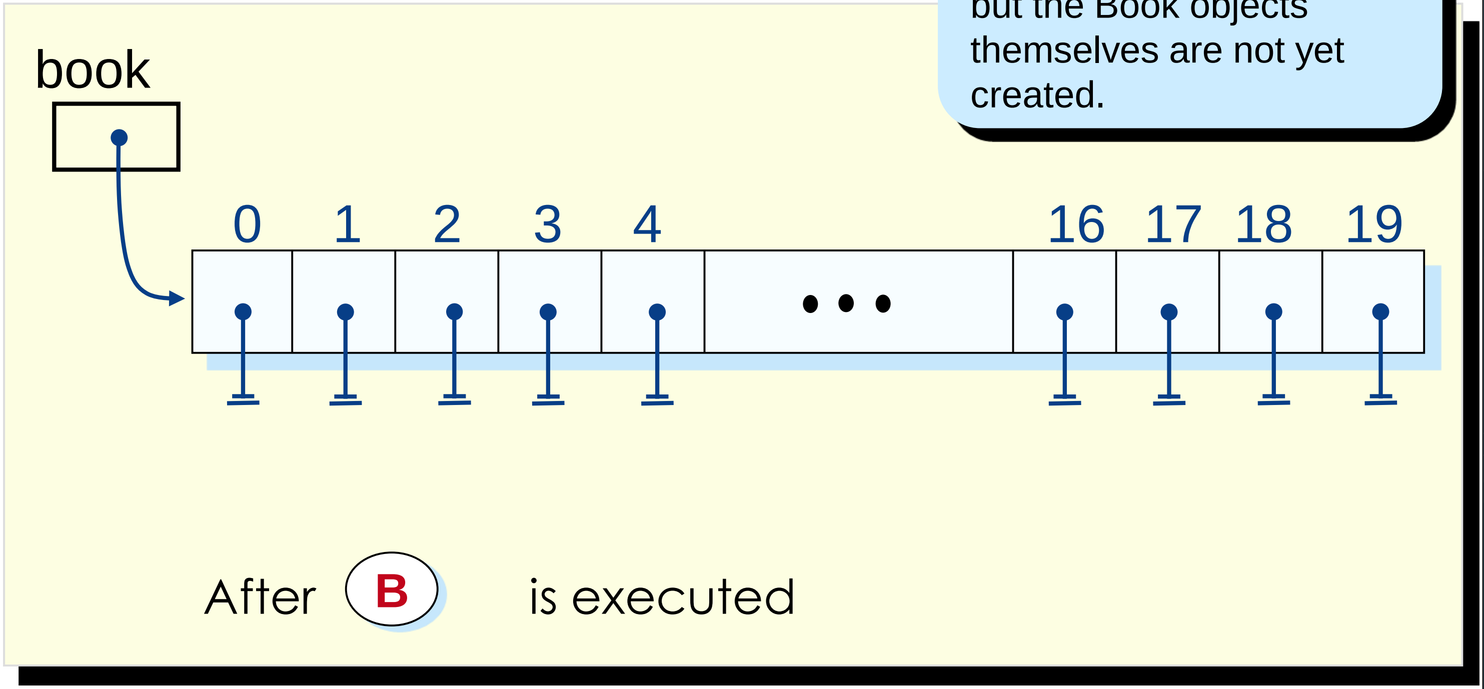
Code

B

```
Book book[];
book = new Book[20];
book[0] = new Book(title, auth, year);
```

Now the array for storing 20 Book objects is created, but the Book objects themselves are not yet created.

State of Memory



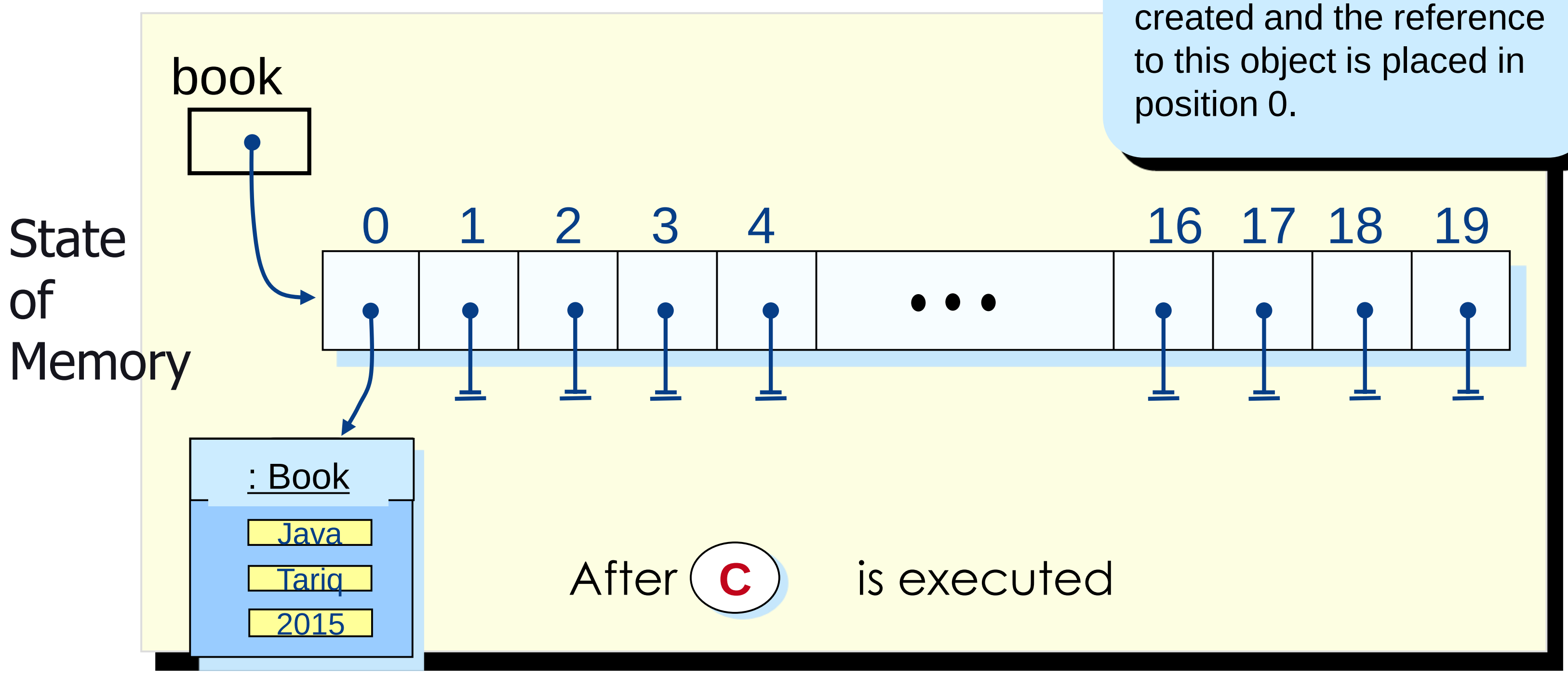
Creating an Object Array - 3

Code

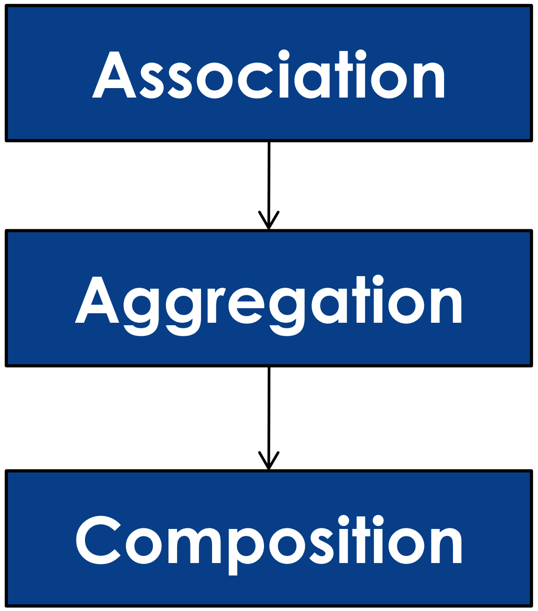
```
Book book[];

book = new Book[20];

book[0] = new Book(title, auth, year);
```

C

One Book object is created and the reference to this object is placed in position 0.



CLASS RELATIONSHIPS



RELATIONSHIPS

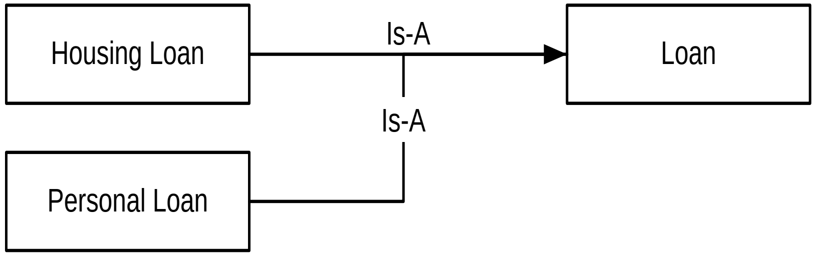
- Different types of relationships can exist between classes
- Identifying relationships helps design the objects better
- There are 3 types of relationships
  - **Is-A (or Kind-Of) [Class-to-Class]**
    - Inheritance
    - Ex: Person - FacultyPerson, StudentPerson, Staff...
    - Ex: ModesOfTravel - Airplane, Train, Auto, Cycle, Boat...
  - **Has-A (or Part-Of) [Object]**
    - Assembly - Parts
    - Group - Members
    - Container - Contents
  - **Uses-A [Object]**
    - FacultyInformation - CourseInformation
    - StudentInformation - CourseInformation

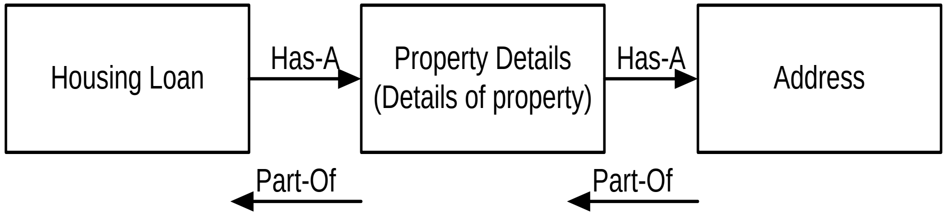
RELATIONSHIPS – CASE STUDY

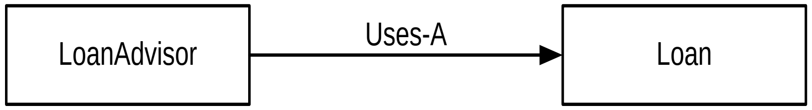
- To understand relationships, let us consider a case study of a banking software
- Global Commerce Bank offers different types of loans
  - Housing Loan
    - Long Term (More than 5 years)
    - Fixed or Floating interest option
    - Documents and details of property to be mortgaged
  - Business Loan
    - Short and Long Term
    - Fixed or Floating interest option
    - Special interest rate can be approved by the Bank Manager
  - Consumer Loan
    - Short Term (Few Months)
    - Fixed interest rate
  - Large Business Loan
    - Short and Long Term
    - Fixed or Floating interest option
    - Special interest rate can be approved by the Bank Manager
    - Moratorium period for repayment

- **Is-A** Relationship (**Inheritance**)
  - **A class is similar to another class**
  - **Class is a different type of another class**
- **Has-A** Relationship (**Aggregation**)
  - **Class contains another class (as member)**
  - **Another class is part of the class**
- **Uses-A** Relationship (**Association**)
  - **Loosely coupled relationship**
  - **A class interacts with another class**

Relationships identified in the case study

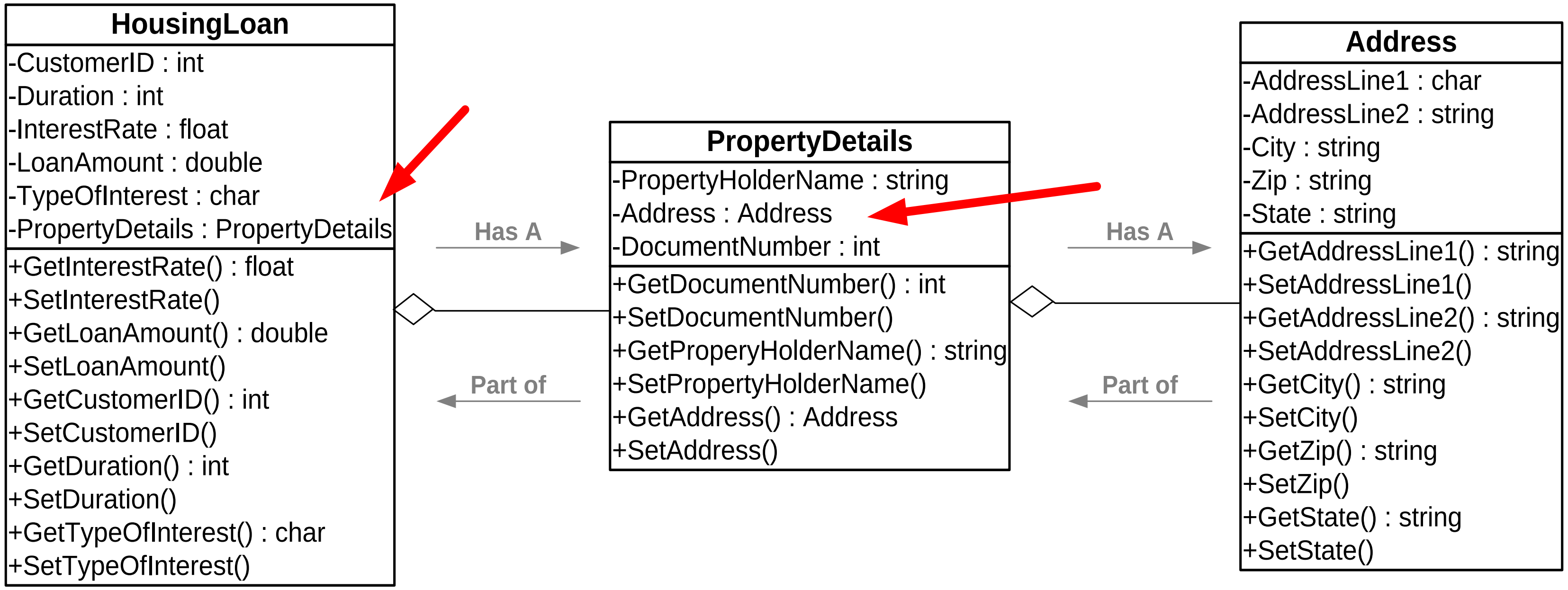






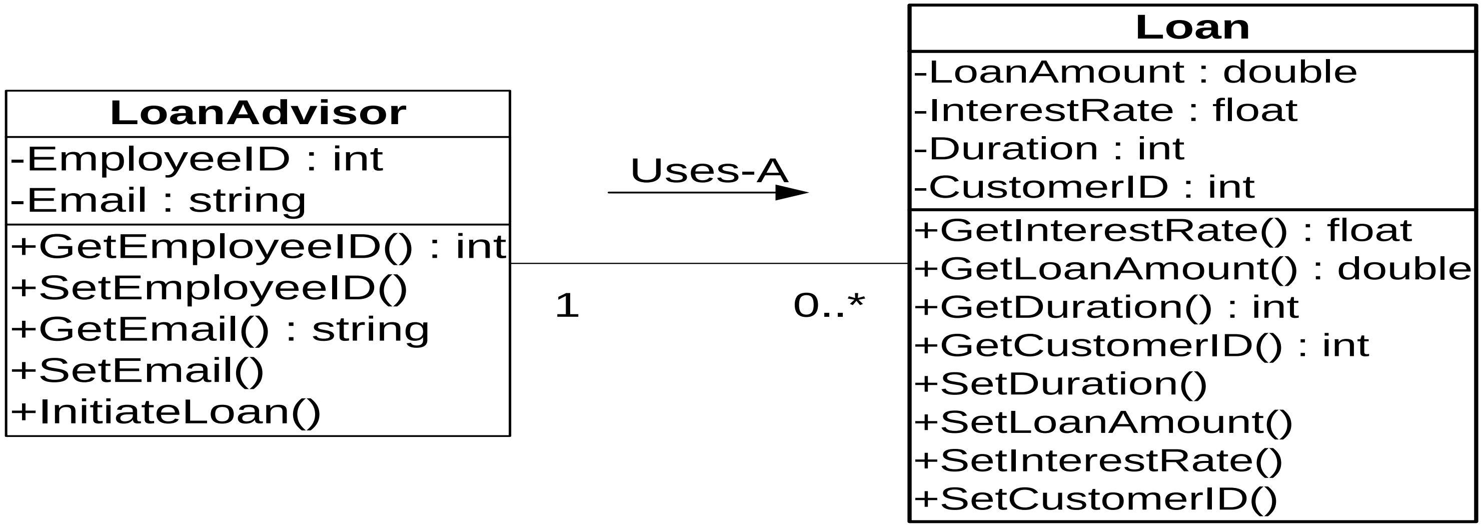
HAS-A RELATIONSHIP - AGGREGATION

- class HousingLoan has 'PropertyDetails' as a member variable
- class PropertyDetails has 'Address' as a member variable
  - Address is a generic class which can store any address (address of a property or address of a person etc)
  - **Has-A relationship is represented with a diamond headed line in UML**



USES-A RELATIONSHIP - ASSOCIATION

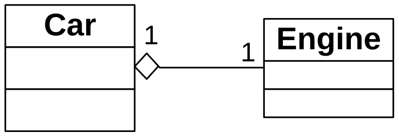
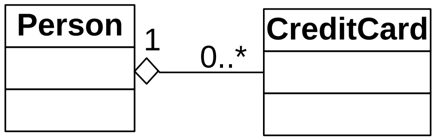
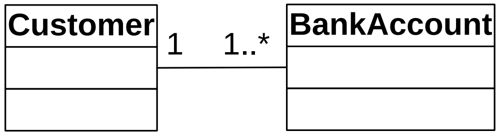
- Objects interacting with other objects. It may include
  - Creation of another type of object
  - Method invocation (Message passing) on already existing object
- Examples:
  - LoanAdvisor creates a Loan object for a new loan
  - LoanAdvisor invokes a method on Loan object



Relationships – Multiplicity of Relationships

Notation	Meaning
1	One only
*	Many (More than one always)
0..1	Zero or One
0..*	Zero or Many
1..*	One or Many

* Applies only to Has-A and Uses-A Relationships

Multiplicity	Representation
One to One Aggregation *A car can have only one engine*	
One to Many Aggregation (Many = zero or more) *A person can have zero or more credit cards.*	
One to Many Association (Many = one or more) *In a bank, a customer can use one or more accounts.*	

Association	Aggregation	Composition
Class A uses Class B.	Class A contains Class B , Or Class A has instance of Class B.	Class A owns Class B.
An association is used when one object wants another object to perform a service for it.	An aggregation is used when life of object is independent of container object But still container object owns the aggregated object.	A composition is used where each part may belong to only one whole at a time.
Life or existence of the associated objects are independent of each other, They just provide some kind of service to each other.	Life or existence of the aggregated objects are independent of each other, But one object is playing the role of Owner of the other object.	Life or existence of the composite object is dependent on the existence of container object, Existence of composite object is not meaningful without its container object.

Characteristic	**Normal Association**	**Aggregation**	**Composition**
UML	———	◇———	◆———
Ownership	None	Weak	Strong
Multiplicity	Any	Any	One : Any
Propagation of properties	Undefined	Whole to Part	Whole to Part

ASSOCIATION

The *association* relationship indicates that a class knows about, and holds a reference to, another class. Associations can be described as a "has-a" relationship because the typical implementation in Java is through the use of an instance field. The relationship can be bi-directional with each class holding a reference to the other. Aggregation and composition are types of association relationships.

ASSOCIATION EXAMPLE

```
public class AntiAirCraftGun {
private Bomber target;
private int positionX;
private int positionY;
private int damage;

public void setTarget(Bomber newTarget)
{      this.target = newTarget;    }   //rest of
AntiAircraftGun class }


public class Bomber {
private AntiAirCraftGun target;
private int positionX;
private int positionY;
private int damage;

public void setTarget(AntiAirCraftGun newTarget)
{      this.target = newTarget;    }   //rest of Bomber
class }
```

AGGREGATION

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

```
class Employee
{
int id;
String name;
Address address;    //Address is a class

}
```

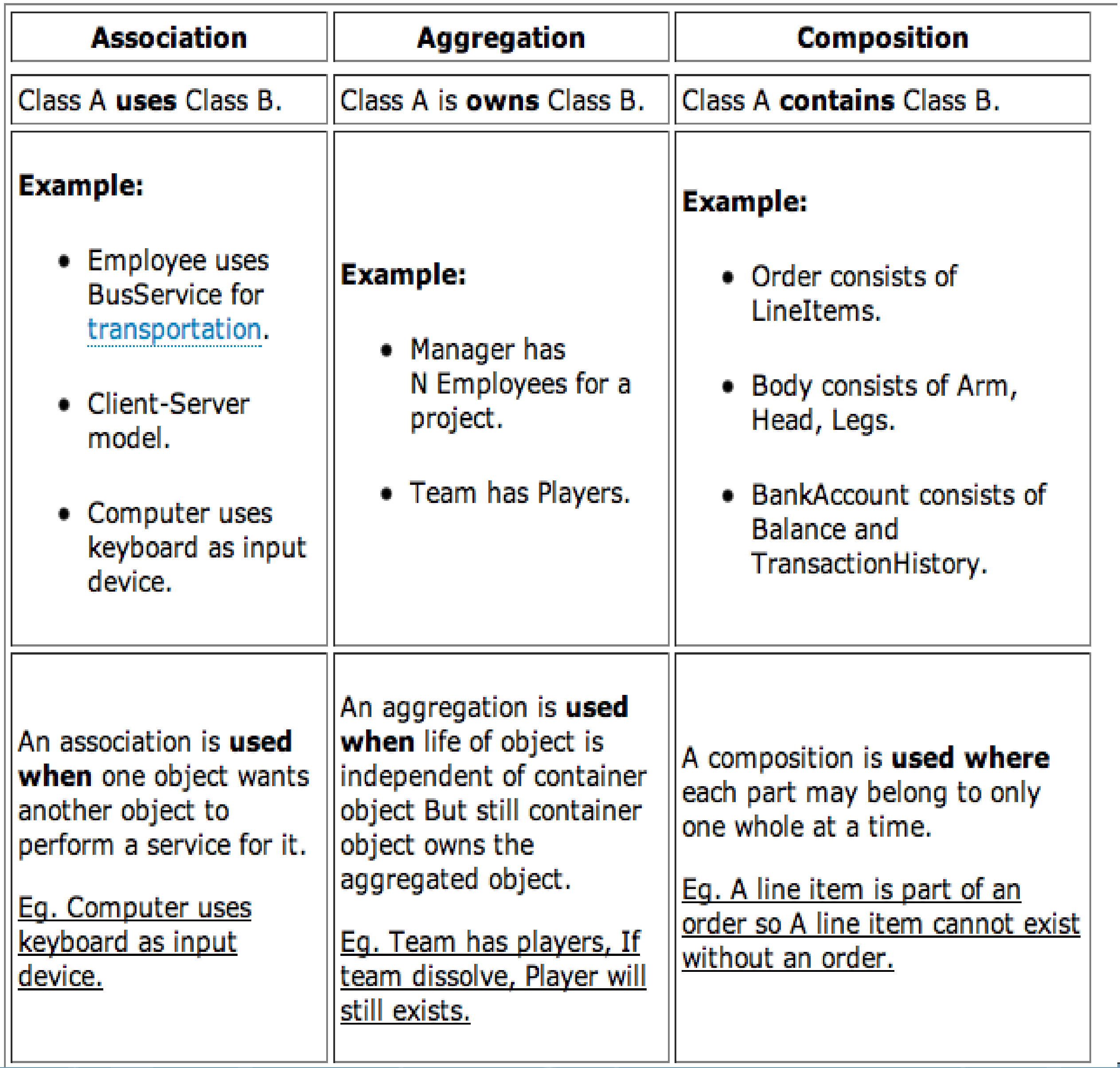
In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

Why use Aggregation?

- For Code Reusability.

Aggregation: weak Has-a of Association

Composition: Strong Has-a of Association



Association	**Aggregation**	**Composition**
Class A **uses** Class B.	Class A is **owns** Class B.	Class A **contains** Class B.
Example: • Employee uses BusService for transportation. • Client-Server model. • Computer uses keyboard as input device.	**Example:** • Manager has N Employees for a project. • Team has Players.	**Example:** • Order consists of LineItems. • Body consists of Arm, Head, Legs. • BankAccount consists of Balance and TransactionHistory.
An association is **used when** one object wants another object to perform a service for it. Eg. Computer uses keyboard as input device.	An aggregation is **used when** life of object is independent of container object But still container object owns the aggregated object. Eg. Team has players, If team dissolve, Player will still exists.	A composition is **used where** each part may belong to only one whole at a time. Eg. A line item is part of an order so A line item cannot exist without an order.

Any Questions