Programming 1

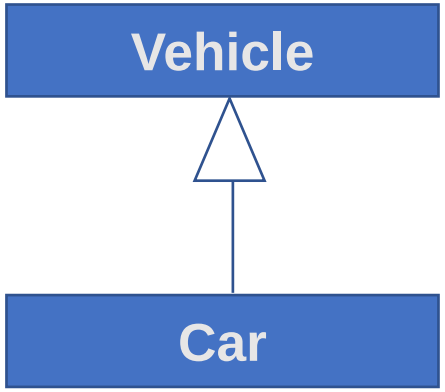
Week 11 – User-defined Classes (Part 3)

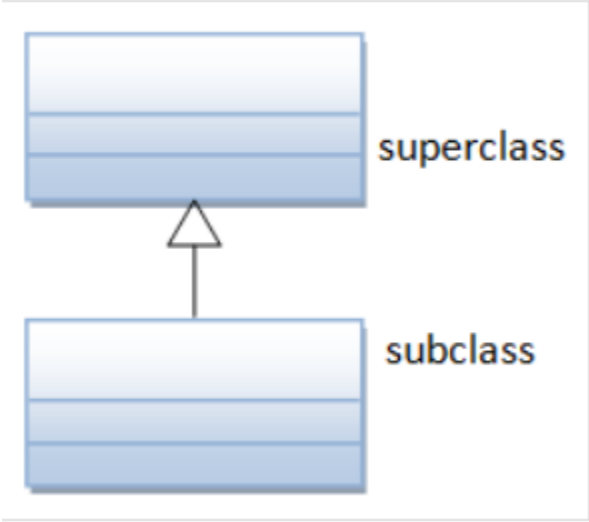
Inheritance

- Another fundamental object-oriented technique is called inheritance, which enhances software design and promotes reuse
  - *Inheritance* allows a software developer to derive a new class from an existing one
  - The existing class is called the *parent class,* or *superclass*, or *base class*
  - The derived class is called the *child class* or *subclass*.
  - As the name implies, the child inherits characteristics of the parent
  - That is, the child class inherits the methods and data defined for the parent class

Inheritance

- Inheritance relationships often are shown graphically in a UML class diagram, with an arrow with an open arrowhead pointing to the parent class. By convention, superclass is drawn on top of its subclasses as shown:





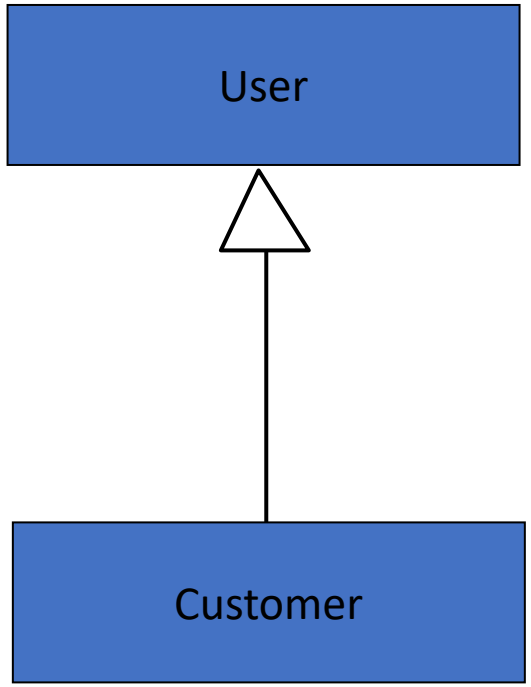
Inheritance should create an *is-a relationship*, meaning the child *is a* more specific version of the parent. *A Car "is a" Vehicle*

Deriving Subclasses

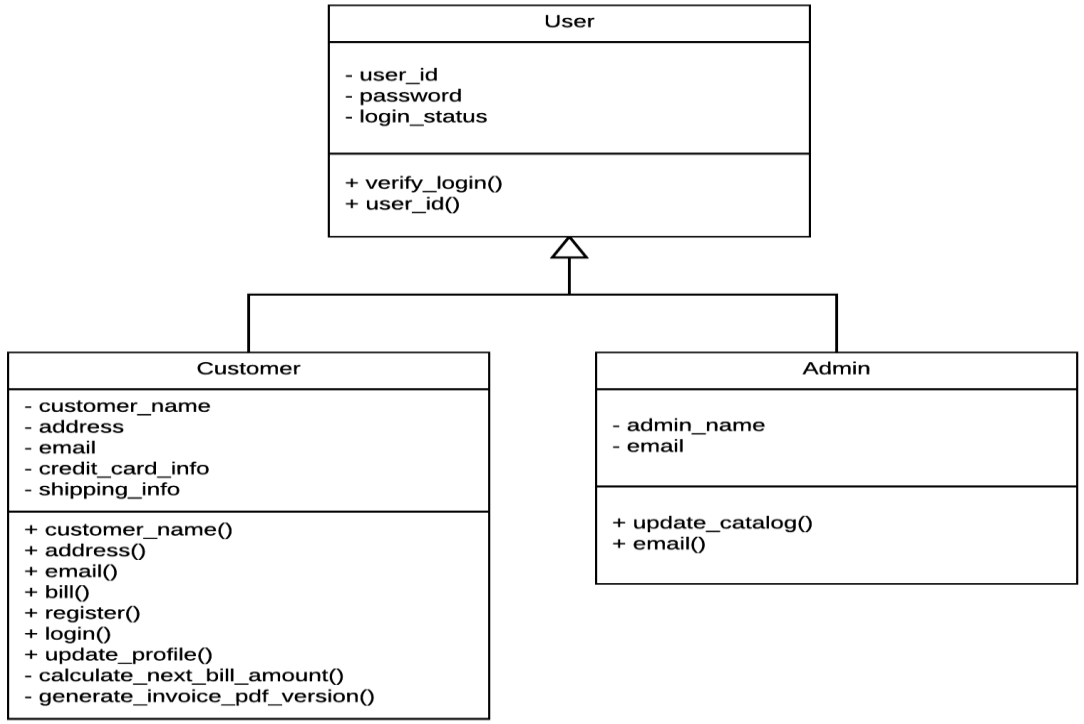
- In Java, we use the reserved word `extends` to establish an inheritance relationship

```
class Car extends Vehicle
{
    // class contents
}
```

Inheritance

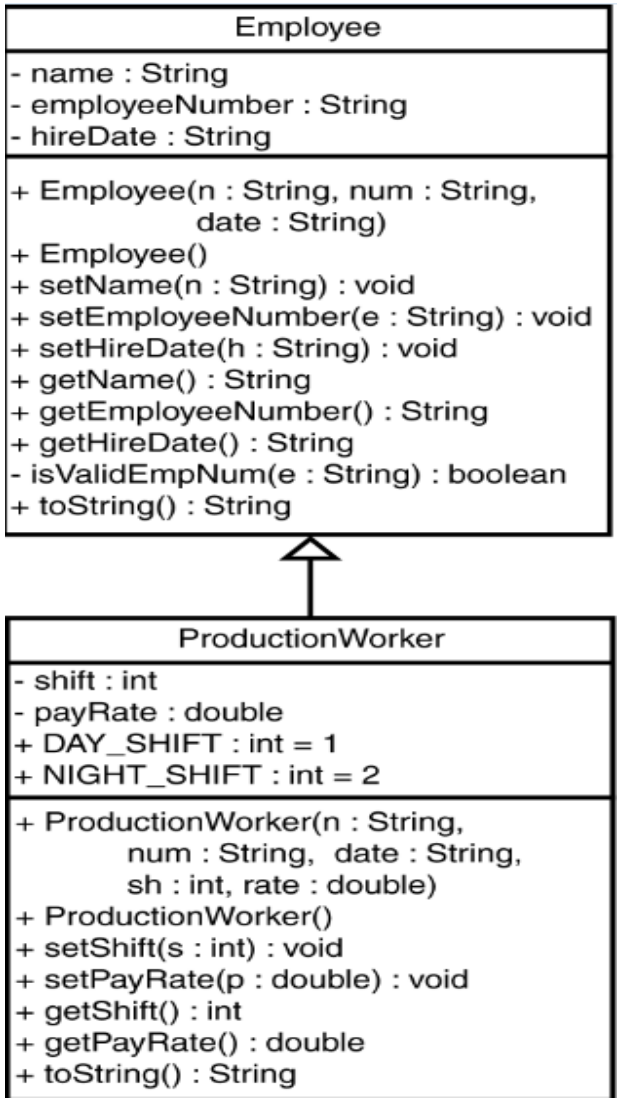


Inheritance (a generalization) relationships connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.



Demo

- The employee class should keep the following information in fields:
- Employee name
- Employee number in the format XXX–L, where each X is a digit within the range 0–9 and the L is a letter within the range A–M.
- Hire date
- Next, write a class named ProductionWorker that extends the Employee class.

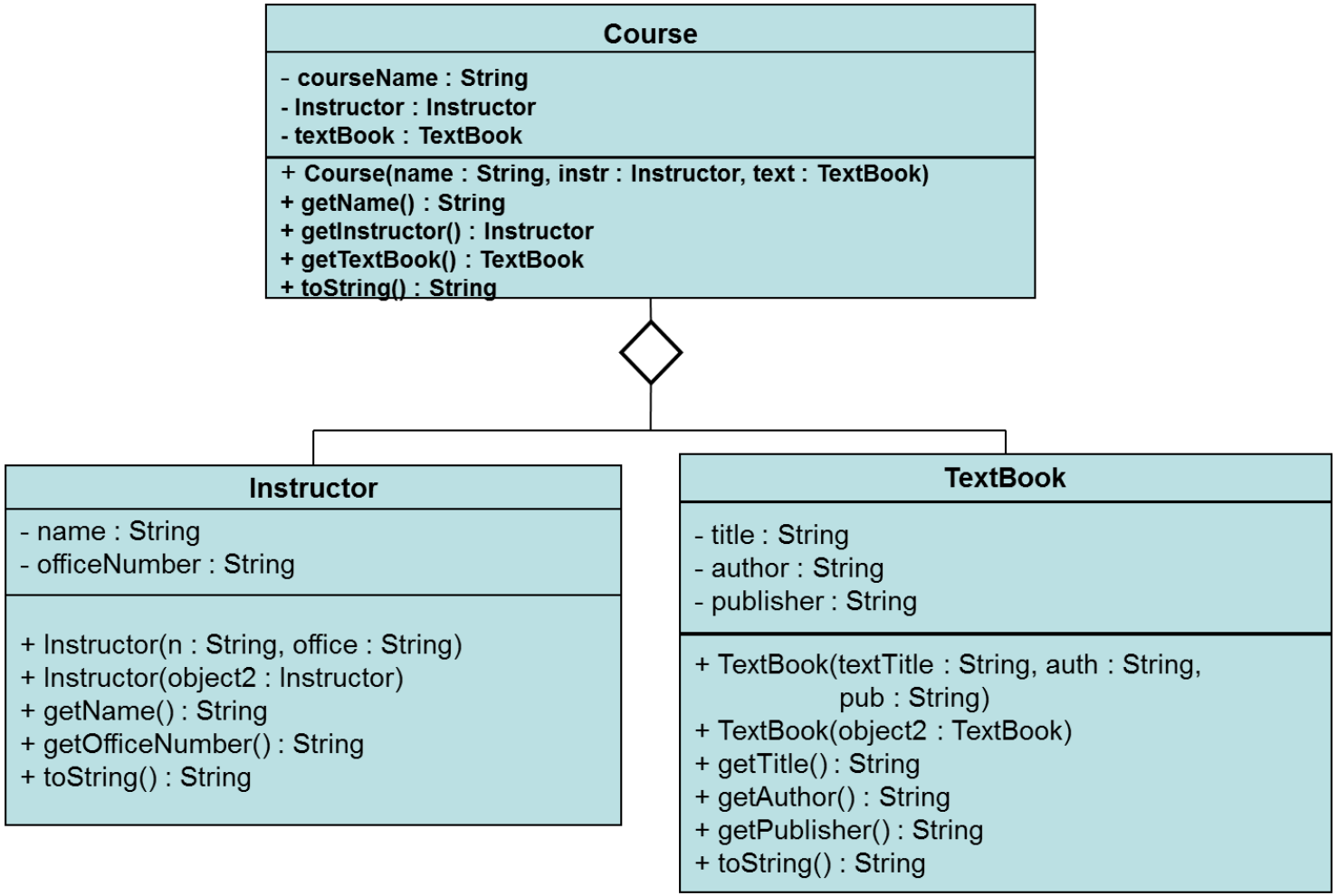


Aggregation

- Not every class relationship is an inheritance relationship.
- Creating an instance of one class as a reference in another class is called **object aggregation**.
- Aggregation creates a “has a” relationship between objects.
- *Has-a* relationship
  - Create classes by composition of existing classes.
  - Example: Given the classes Employee, BirthDate and TelephoneNumber, it’s improper to say that an Employee *is a* BirthDate or that an Employee *is a* TelephoneNumber.
  - However, an Employee *has a* BirthDate, and an Employee *has a* TelephoneNumber.

Aggregation in UML Diagrams

- **Aggregation** implies a relationship where the child can exist independently of the parent. Example: Class (Course) and Instructor(child). Delete the Course and the Instructor still exist.

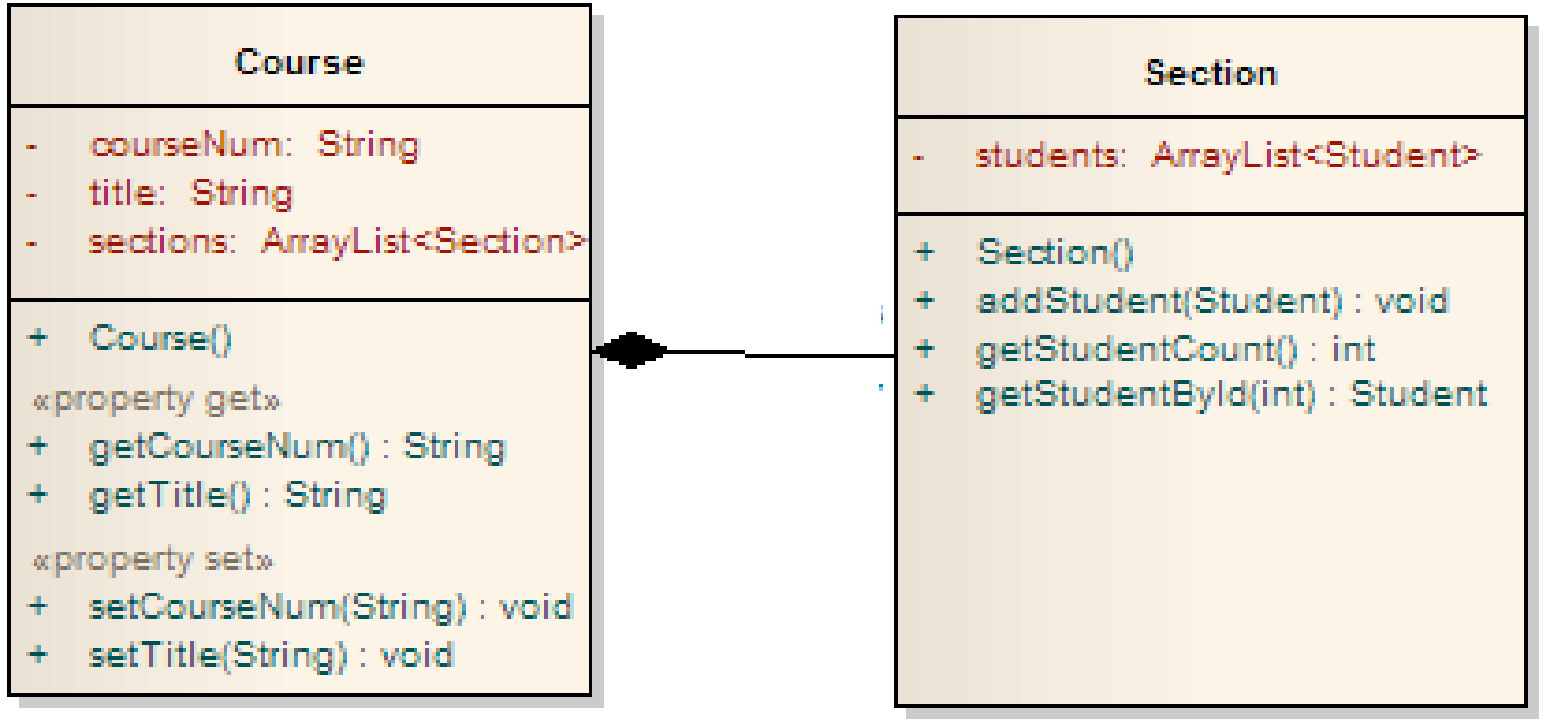


Composition in UML Diagrams

A *composition* models the part-whole relationship. Every part may belong to only one whole, and if the whole is deleted, so are the parts
Compositions are denoted by a filled-diamond adornment on the association.

Composition in UML Diagrams

Composition implies a relationship where the child cannot exist independent of the parent. Example: Course (parent) and Section(child). Section don't exist separate to a Course.



Demo

Enumerated Types (1 of 2)

- Known as an enum, requires declaration and definition like a class
- An enum is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).
- Syntax:

```
enum typeName { one or more enum constants }
```

- Definition:

```
enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
           FRIDAY, SATURDAY }
```

- Declaration:

```
Day WorkDay; // creates a Day enum
```

- Assignment:

```
Day WorkDay = Day.WEDNESDAY;
```

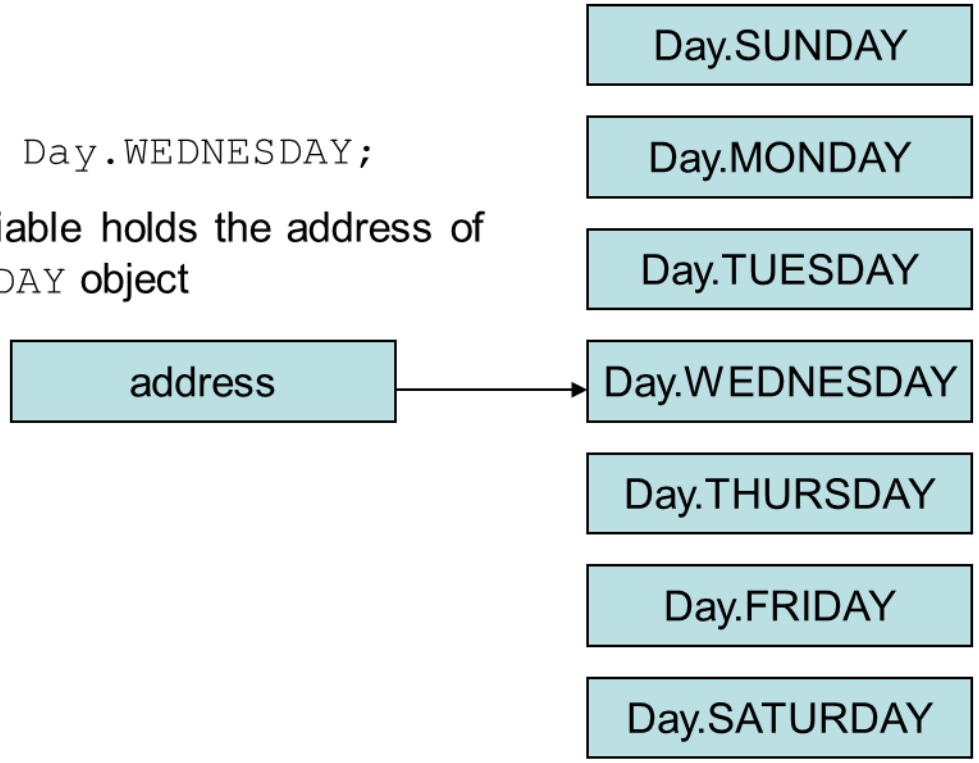
Enumerated Types (2 of 2)

- An `enum` is a specialized class

Each are objects of type `Day`, a specialized class

```
Day workDay = Day.WEDNESDAY;
```

The `workDay` variable holds the address of the `Day.WEDNESDAY` object



Enumerated Types - Switching

- Java allows you to test an enum constant with a `switch` statement.

Why And When To Use Enums?

- Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

Garbage Collection (1 of 6)

- When objects are no longer needed, they should be destroyed.
- so the memory it uses can be freed for other purposes
- Java handles all of the memory operations for you.
- Simply set the reference to **null** and Java will reclaim the memory.

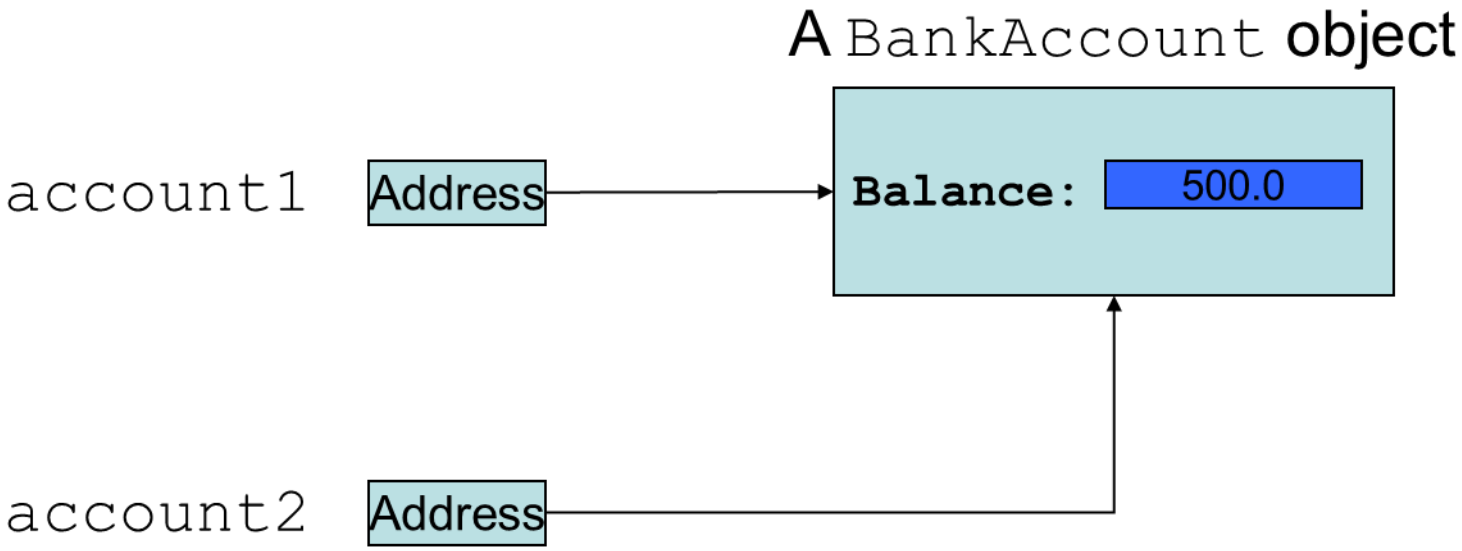
Garbage Collection (2 of 6)

- The Java Virtual Machine has a process that runs in the background that reclaims memory from released objects
- The **garbage collector** will reclaim memory from any object that no longer has a valid reference pointing to it.

```
BankAccount account1 = new BankAccount(500.0);
BankAccount account2 = account1;
```

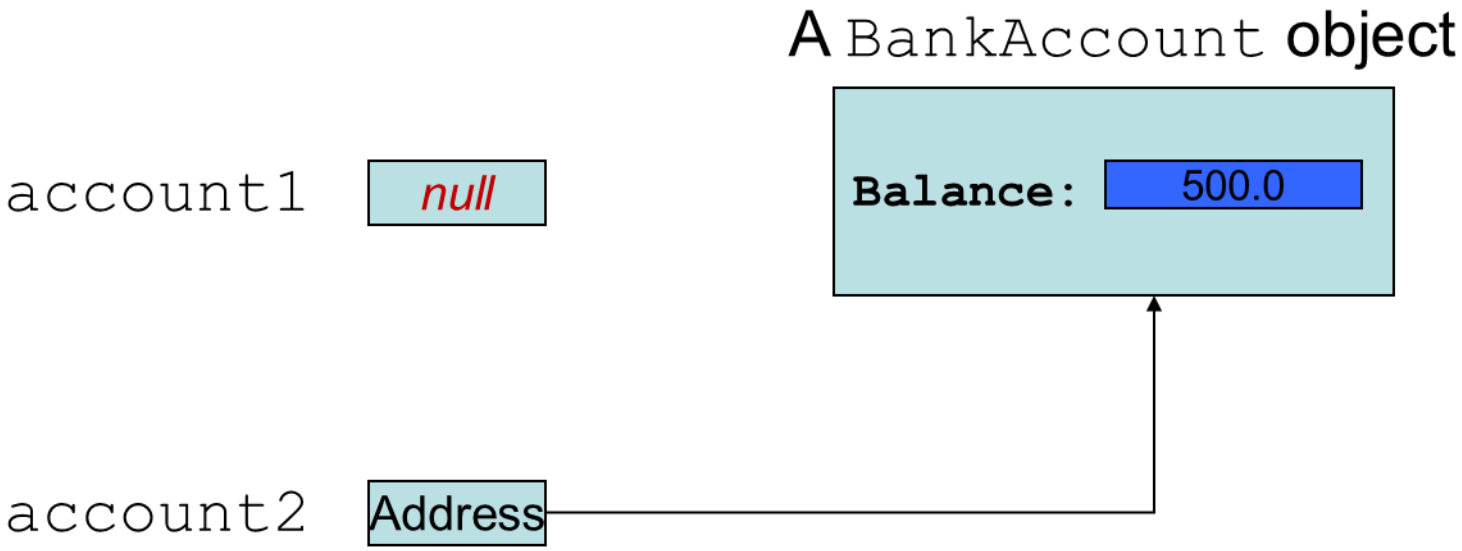
- This sets `account1` and `account2` to point to the same object.

Garbage Collection (3 of 6)



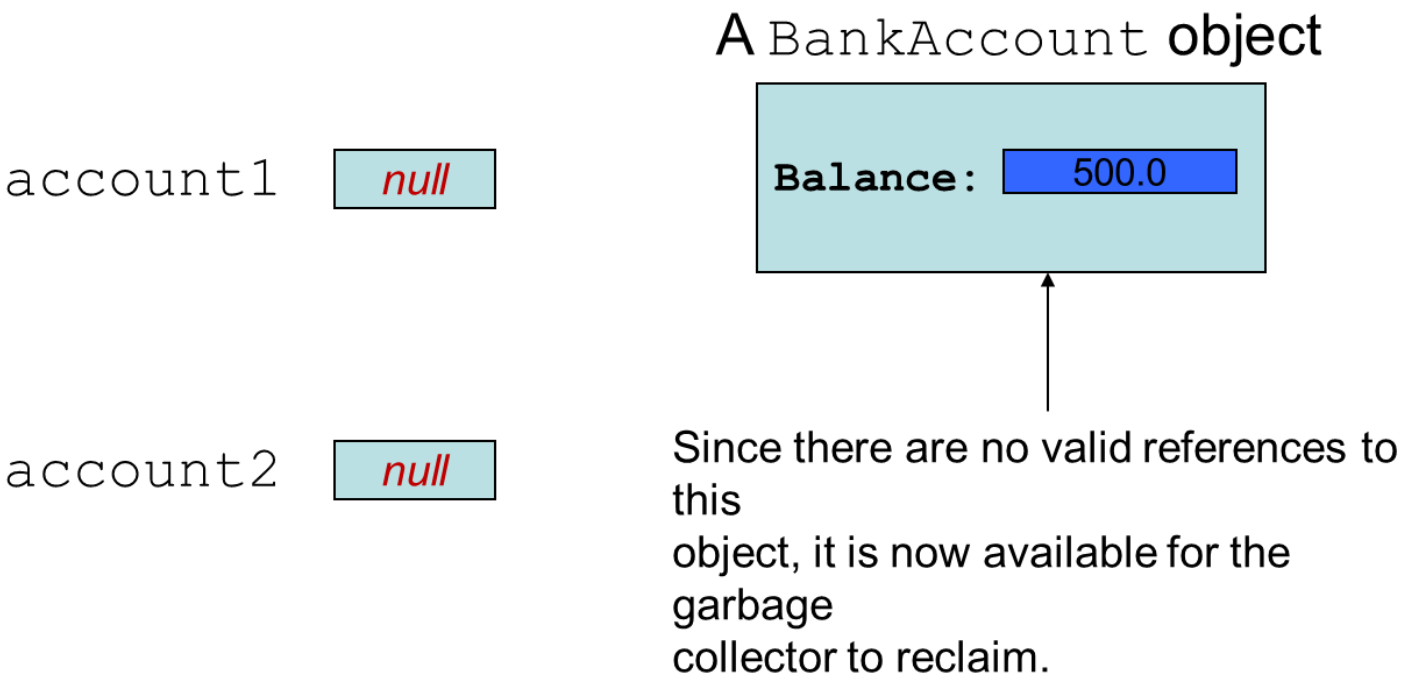
Here, both `account1` and `account2` point to the same instance of the `BankAccount` class.

Garbage Collection (4 of 6)



However, by running the statement: **`account1 = null;`**
only `account2` will be pointing to the object.

Garbage Collection (5 of 6)



If we now run the statement: **`account2 = null;`**
neither `account1` or `account2` will be pointing to the object.

Garbage Collection (6 of 6)