JAVA
Programming

Aggregation, Composition
and inner classes

Overview

- Visibility modifiers
- Aggregation and Composition
- Factories
- Nested Types
    - Inner classes
    - Local classes
    - Anonymous classes

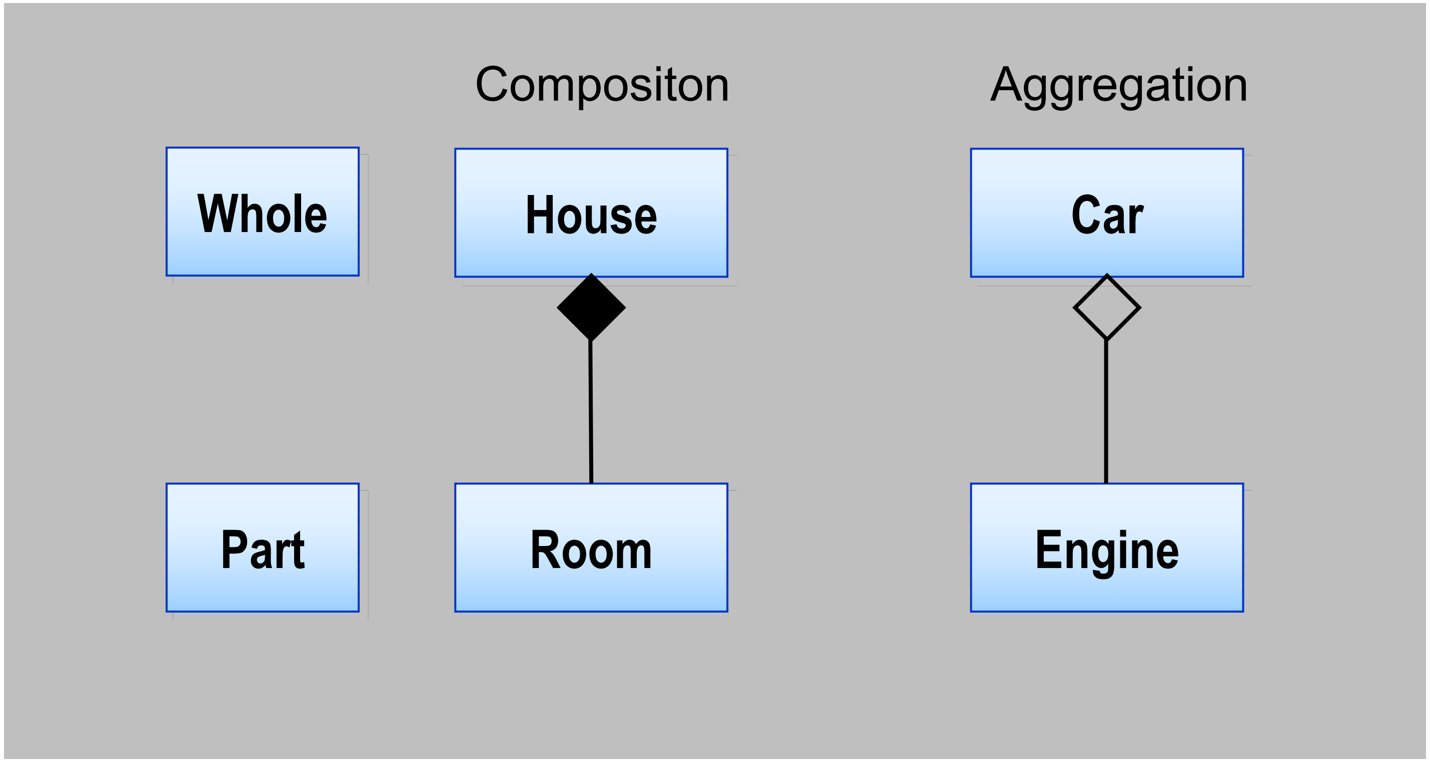
Visibility modifiers

- public
  - accessible to all
- protected
  - accessible to derived classes and classes in same package
- private
  - accessible to current class only
- No modifier
  - Accessible by classes in same package

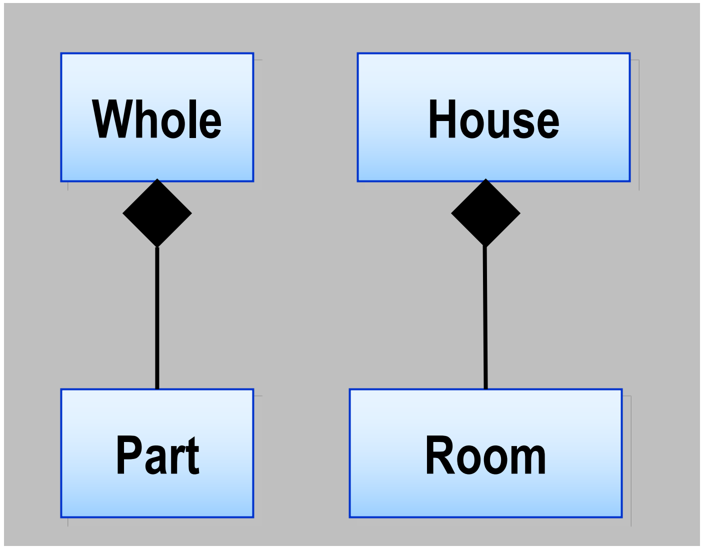
Aggregation and Composition

- Inheritance gives a “is-a” relationship
- Composition gives a “is-part-of” relationship, in which the lifetime of the part is managed by the whole
- Aggregation gives a “has-a” relationship, in which the lifetime of the part is not controlled by the whole

Aggregation and Composition



Composition

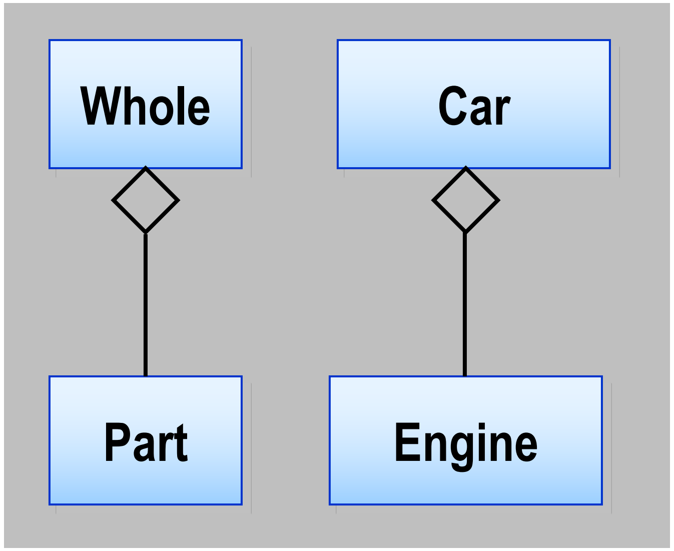


```
public class House {

  private Room masterBedRoom;

  public House() {
        masterBedRoom=new Room();
  }
}
```

Aggregation



```
public class Car {

   private Engine engine;

   public Car(Engine engine) {
           this.engine=engine;
   }
}
```

```
public static void main(String[] args) {
   Engine engine=new Engine();
   Car c=new Car(engine);
}
```

Factories

- Creation is often complex and restricted
- Many objects are made only in specialist factories
- The factory encapsulates the complex creation
- Factories are useful patterns when modelling software

Factory Example

```
public class Bank {

public enum AccountType { checking,saving
};

public Bank() {
}

public BankAccount createAccount(AccountType accountType) {
  switch (accountType) {
    case checking: return new CheckingAccount();
    case saving: return new SavingAccount();

   }
  }
}
```

Factory Example(cont)

```
public interface BankAccount {
  long getAccountNumber();

  void setAccountNumber(long accountNumber);

  int getBalance();

  void setBalance(int balance);

}
```

Factory Example (cont)

```
class CheckingAccount implements BankAccount {
  private long accountNumber;
  private int balance;

  public long getAccountNumber() {
    return accountNumber;
  }

  public void setAccountNumber(long accountNumber) {
    this.accountNumber = accountNumber;
  }

  public int getBalance() {
    return balance;
  }

  public void setBalance(int balance) {
    this.balance = balance;
  }

public CheckingAccount() {
}
}
```

Factory Example (cont)

```java
public static void main(String[] args) {

   Bank bank = new Bank();
   BankAccount newCheckingAccount = bank
         .createAccount(Bank.AccountType.checking);
   BankAccount newSavingAccount = bank
         .createAccount(Bank.AccountType.saving);
}
```

Nested Types

- Classes/Interfaces declared in other Classes/Interfaces
- Static and non-static nested types
- Non-static nested classes are called Inner Classes

Nested Types

```
public class EnclosingClass {

  public static class StaticNestedClass {
       // ....
  }

  public class InnerClass {
       // ....
  }
}
```

```
// create instance of static nested class
EnclosingClass.StaticNestedClass staticNestedObject = new
       EnclosingClass.StaticNestedClass();
// create instance of InnerClass
EnclosingClass enclosingObject = new EnclosingClass();
EnclosingClass.InnerClass innerObject = enclosingObject.new
                                            InnerClass();
```

Inner Classes

- Inner Classes can access members of enclosing class
- Static nested classes have no access to members of enclosing class
- Inner Interfaces are implicitly static

Nested Types

```
public class EnclosingClass {
  private int count=10;
  private static String data="abc";

  public static class StaticNestedClass {
    public StaticNestedClass(){
        count++; //count not accessible here
        data.toUpperCase();//ok
    }
  }

  public class InnerClass {
    public InnerClass(){
        count++;//ok
        data.toUpperCase();//ok
    }
  }
}
```

Local Classes

- Defined in any code block like method body, constructor, initialization block
- Not member of enclosing class, but local to the block in which they are defined
- Local variable or method parameter only accessible if declared final

Local Classes

```java
@Override
public Iterator<Integer> iterator() {
  class LinkedListIterator implements Iterator<Integer> {
    private Node currentNode = head;
    @Override
    public boolean hasNext() {
      return currentNode != null;
    }
    @Override
    public Integer next() {
      Integer value= currentNode.getValue();
      currentNode=currentNode.getNext();
      return value;
    }
    @Override
    public void remove() {
    }
  }
  return new LinkedListIterator();
}
```

```java
for (Integer value : linkedList) {
		System.out.println(value);
}
```

Anonymous Classes

- Declared without a name
- Defined in the new expression itself
- No explicit extends or implements clause allowed
- No modifiers or annotations

Anonymous Classes

```java
public Iterator<Integer> iterator() {
  return new Iterator<Integer>() {
    private Node currentNode = head;

    @Override
    public boolean hasNext() {
      return currentNode != null;
    }

    @Override
    public Integer next() {
      Integer value= currentNode.getValue();
      currentNode=currentNode.getNext();
      return value;
    }

    @Override
    public void remove() {
    }
  };
}
```

Lab: Using inner classes