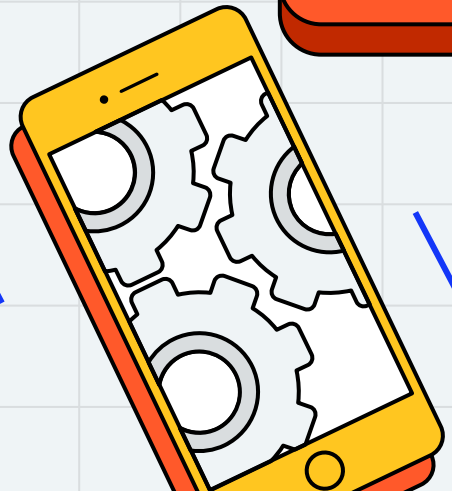
Application Programming

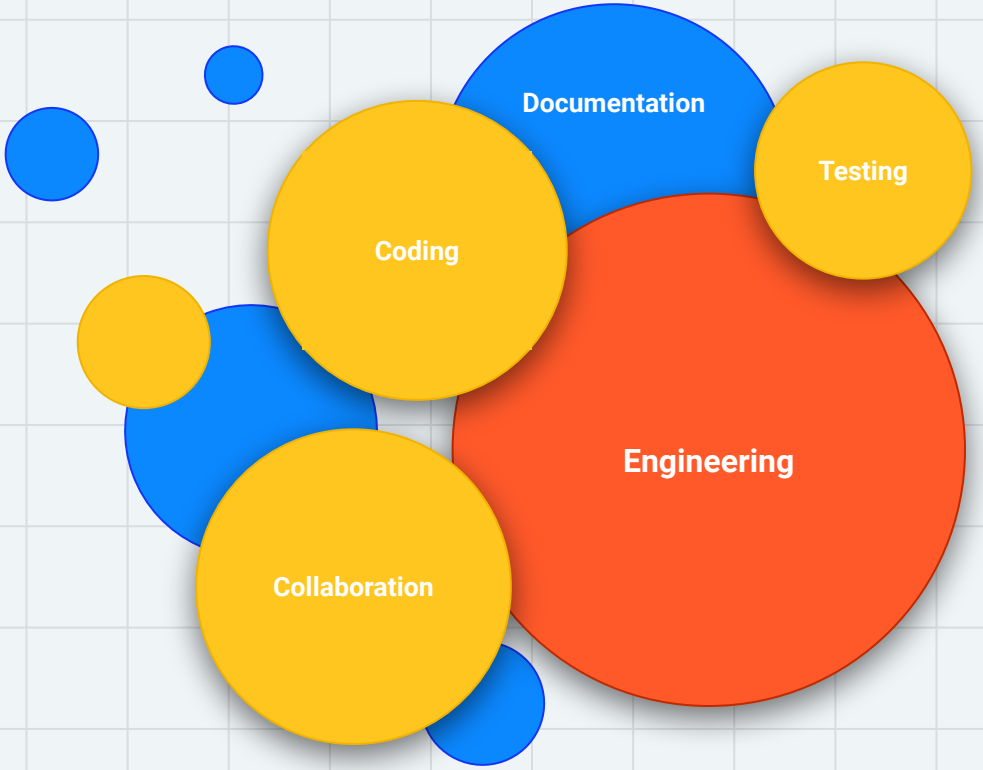


Hend Alkittawi

UML Diagrams

UML Diagrams and Class Relationships

WHAT IS APPLICATION PROGRAMMING?



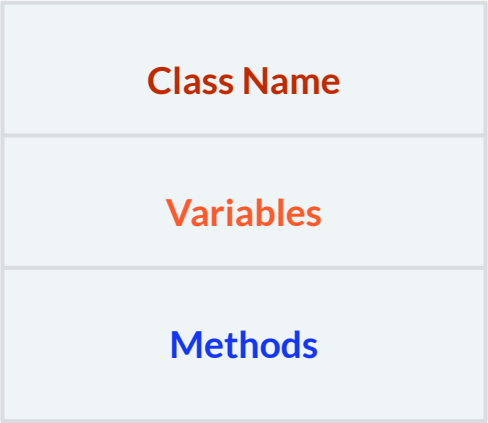
UML DIAGRAMS

- UML: **U**nified **M**odeling **L**anguage
  - Graphical models of object-oriented software

- What do you think such graphical models should include?

UML DIAGRAMS

- UML: **U**nified **M**odeling **L**anguage
  - Graphical models of object-oriented software



UML DIAGRAMS

- Create the UML diagram for the `Account.java` class

```
public class Account {
    private String name;
    private double balance;

    public Account(String name) {// method body here}
    public void setName(String name) {// method body here}
    public void setBalance(double bal) {// method body here}
    public String getName() {// method body here}
    public double getBalance() {// method body here}
}
```

Class Name
Variables
Methods

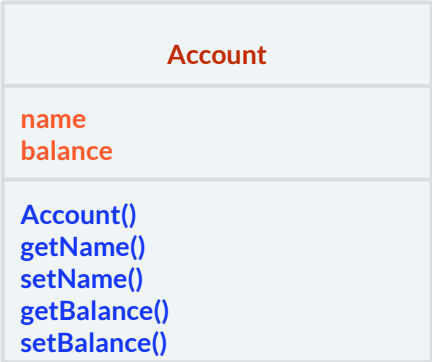
UML DIAGRAMS

- Create the UML diagram for the `Account.java` class

```
public class Account {

    private String name;
    private double balance;

    public Account(String name, double balance) {// method body here}
    public void setName(String name) {// method body here}
    public void setBalance(double bal) {// method body here}
    public String getName() {// method body here}
    public double getBalance() {// method body here}
}
```



UML DIAGRAMS

- Create the UML diagram for the `Account.java` class
  - A <<interface>> tag indicates interface class
  - A <<constructor>> tag indicates constructor
  - Include variables data types
  - Include method parameters and their data type
  - Include method return data type
  - An underscore (_) indicates static
  - A minus(-) indicates private
  - A plus (+) indicates public
  - A hash (#) indicates protected

Account

- name: String
- balance: double

<<constructor>> Account(name: String, balance: double)
+ getName() : String
+ setName(name: String)
+ getBalance() : double
+ setBalance(amount: double)

UML DIAGRAMS

- Create the UML diagram for the `Account.java` class

```java
public class Account {
  private String name;
  private double balance;

  public Account(String name, double balance) {// method body here}
  public void setName(String name) {// method body here}
  public void setBalance(double bal) {// method body here}
  public String getName() {// method body here}
  public double getBalance() {// method body here}
}
```

Account
- name: String - balance: double
<<constructor>> Account(name: String, balance: double) + getName() : String + setName(name: String) + getBalance() : double + setBalance(amount: double)

UML Diagrams

- Classes depend and interact with each other

- UML diagrams can visually show the interactions and dependencies between classes

- Can you name relationships/dependencies between classes?

JAVA CLASSES

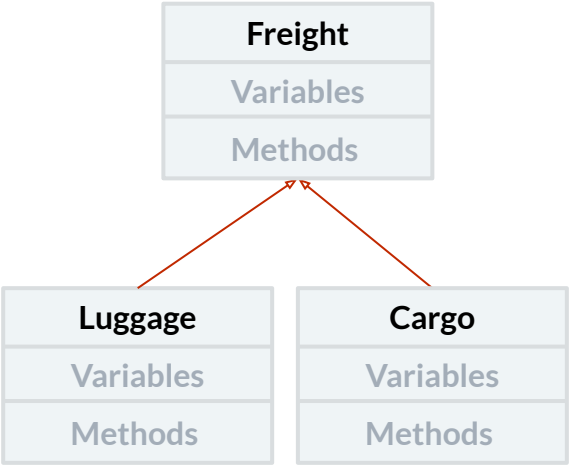
- In a Java application classes are related to each other
  - In an **is-a relationship** an object of a subclass can also be treated as an object of its superclass, or an object that implements an interface can be treated as an object of the interface (**polymorphism**).
    - Example: A SavingsAccount class that extends Account class
    - Example: An Invoice class that implements Payable interface
  - In a **has-a relationship**, an object contains as members references to other objects (**composition**).
    - Example: An Employee class where an employee has one or more Date objects
    - Example: A Bank class where a bank has one or more Account objects

UML Diagrams

- Dependencies and relationships between classes can be
  - Dependency relationship
  - Unidirectional association
  - Bidirectional association
  - Aggregation relationship
  - Composition relationship
  - Realization relationship
  - Generalization relationship

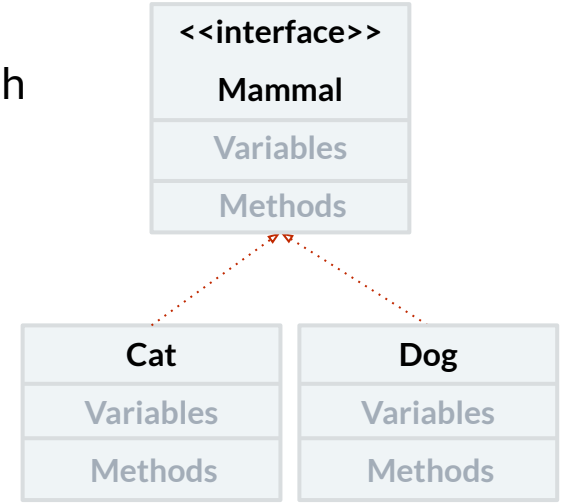
UML DIAGRAMS

- Generalization relationship
  - indicates **inheritance** between classes
- *ClassA* has a generalization relationship with *ClassB* implies
  - *ClassA* inherits from *ClassB*
  - *ClassA* is the subclass, and *ClassB* is the superclass
  - declared in code as *ClassA* **extends** *ClassB*



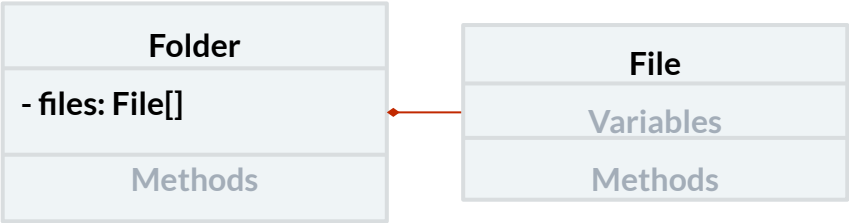
UML DIAGRAMS

- Realization relationship
  - indicates **inheritance**, where *ClassB* is an interface
- *ClassA* has a realization relationship with *ClassB* implies
  - *ClassA* inherits from *ClassB*
  - declared in code as *ClassA* **implements** *ClassB*



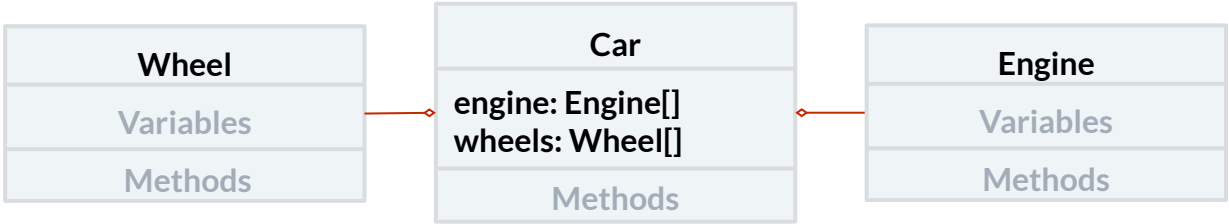
UML DIAGRAMS

- Composition relationship
  - indicates **not-shared association**
  - **strong dependency**
- *ClassA* has a composition relationship with *ClassB* implies
  - *ClassA* is a container for a **data structure** of *ClassB*
  - *there is not a super-sub / parent-child relationship between ClassA and ClassB*
  - if *ClassA* is deleted, *ClassB* need to be removed as well



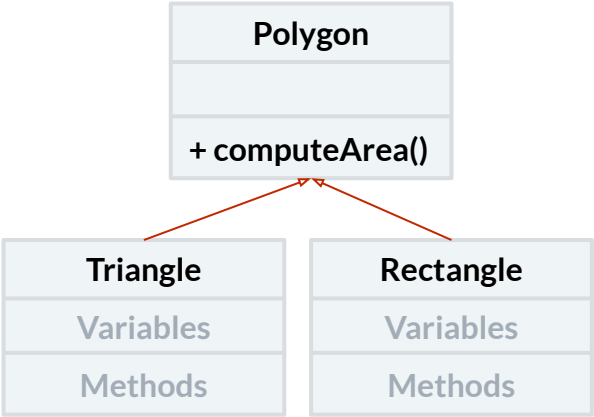
UML DIAGRAMS

- Aggregation relationship
  - indicates **shared association**
- *ClassA* has an aggregation relationship with *ClassB* implies
  - *ClassA* references a **data structure** of *ClassB*, but is not the only reference
  - *ClassA* does not own *ClassB*
  - there is not a *super-sub / parent-child* relationship between *ClassA* and *ClassB*



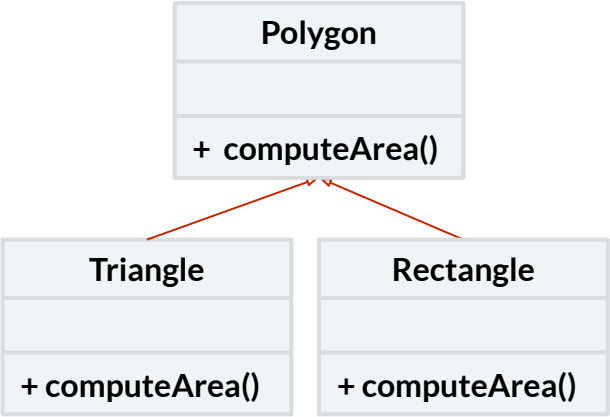
CLASS ACTIVITY

- What does this UML tell you about the corresponding Java code?



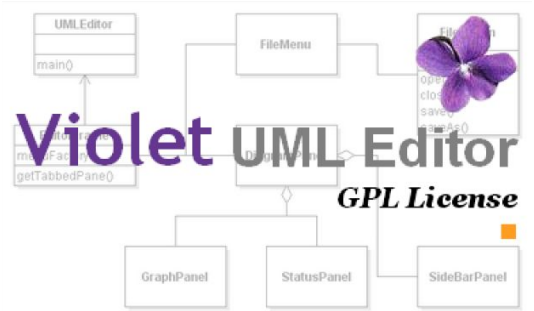
CLASS ACTIVITY

- What does this UML tell you about the corresponding Java code?
  - We have three Java classes
  - Polygon is an abstract class
  - computeArea() is an **abstract** method
  - A Rectangle is-a Polygon
    - Rectangle extends Polygon
  - A Triangle is-a Polygon
    - Triangle extends Polygon

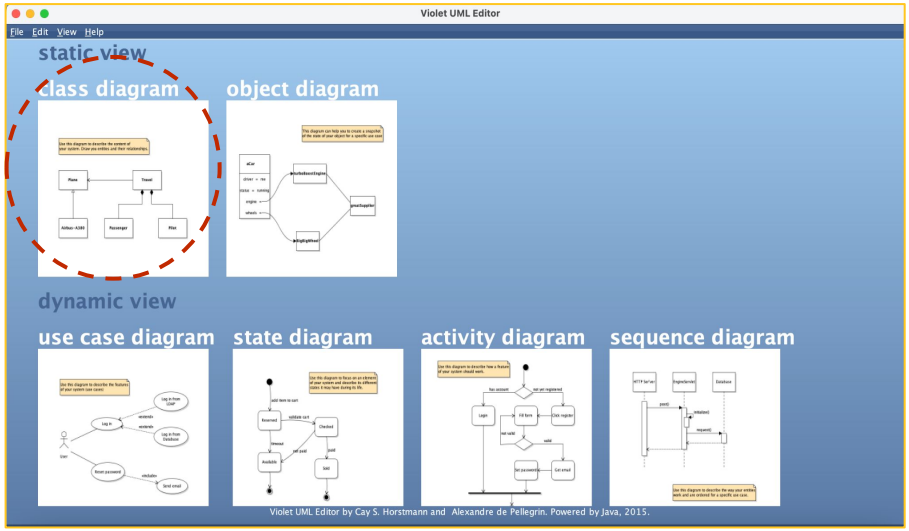


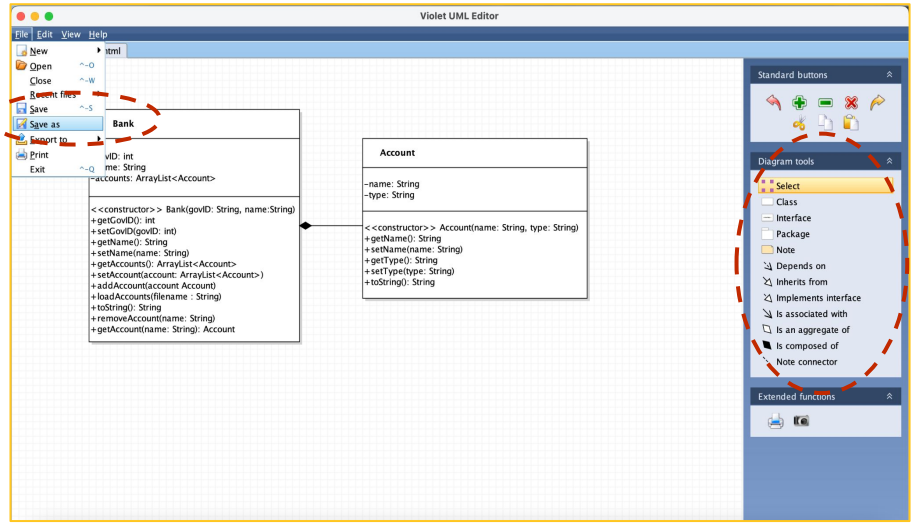
VIOLET UML EDITOR

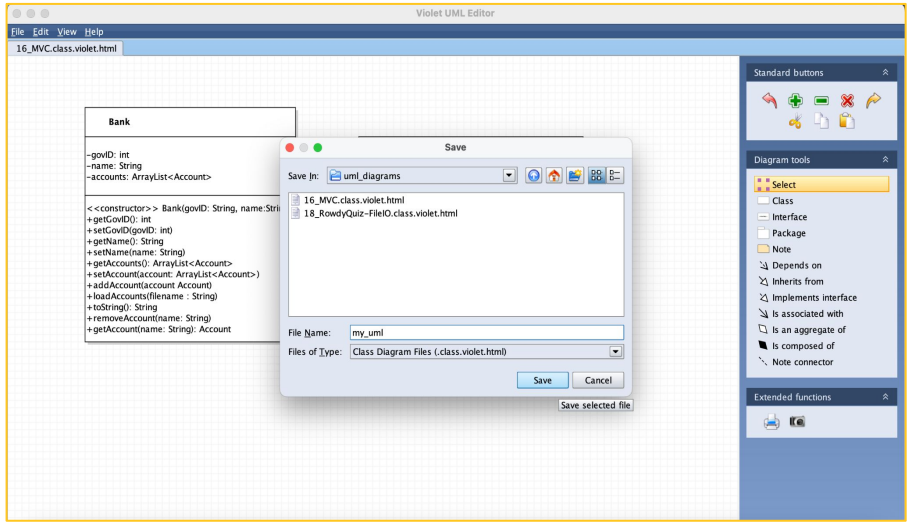
- A free, cross-platform UML editor
  - You can get it here
- We will be creating **Class Diagrams**
- The UML diagram is saved as an html file and can be exported as an image (*. png) file



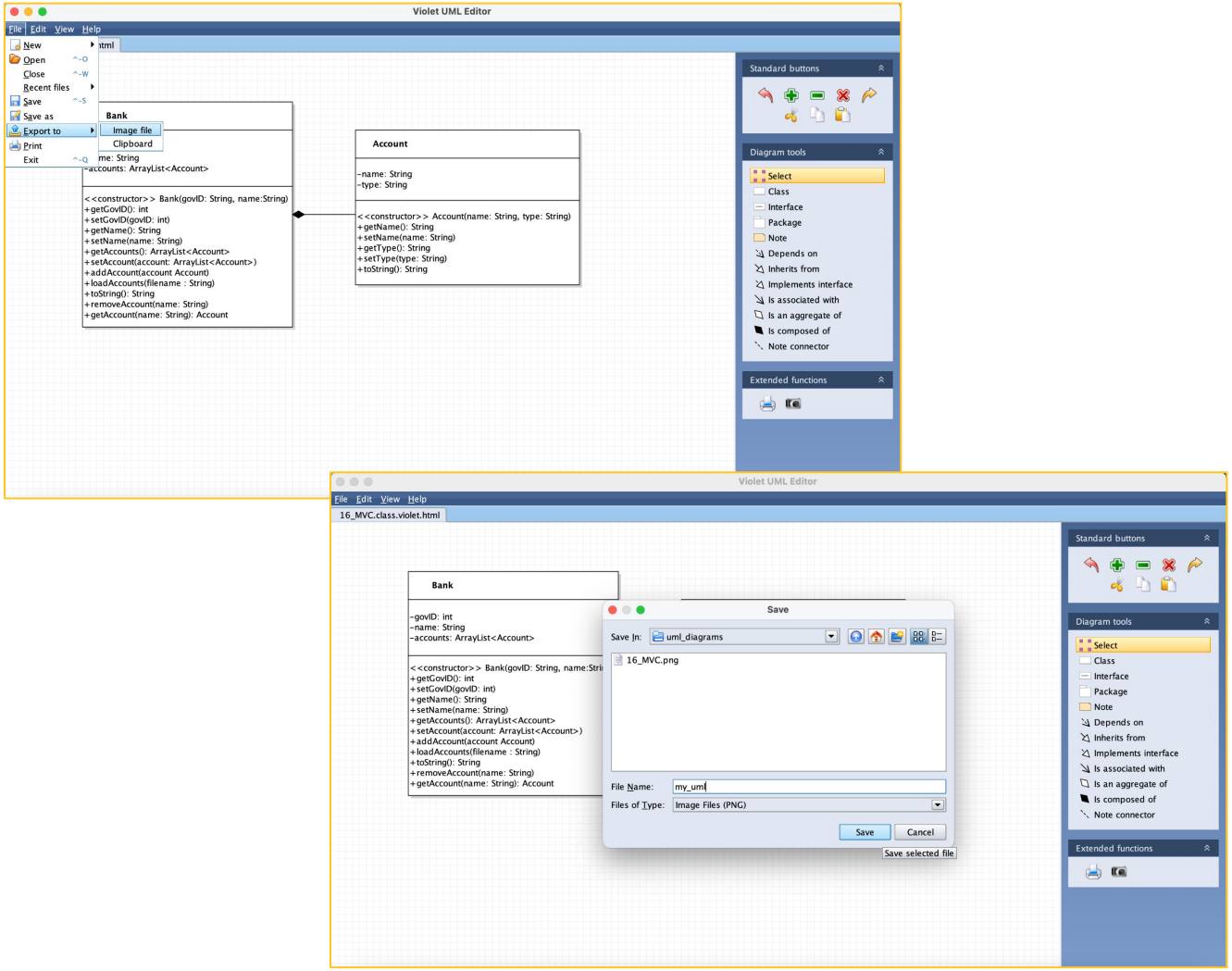
Save As *.html file







Save As *.png file

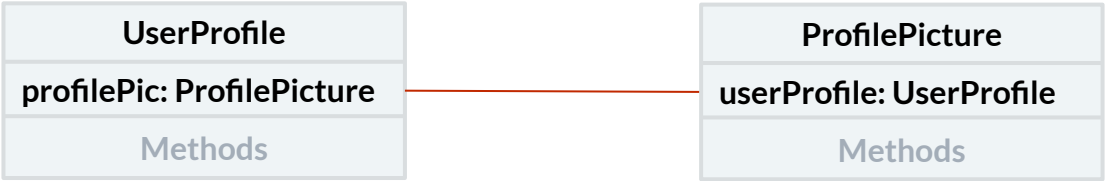


UML DIAGRAMS

- Unidirectional association
  - indicates **dependency** of only one class on another
- *ClassA* has a unidirectional association with *ClassB* implies
  - *ClassA* uses and depends on *ClassB*, but *ClassB* does not reference *ClassA*
  - at least one *ClassB* object is referenced in *ClassA*
  - there is an import statement in *ClassA* for *ClassB*
  - there is not an import *ClassA* statement in *ClassB*

UML DIAGRAMS

- Bidirectional association
  - indicates **codependency** of two classes
- *ClassA* has a bidirectional association with *ClassB* implies
  - both depend upon each other
  - at least one *ClassB* object is referenced in *ClassA*
  - at least one *ClassA* object is referenced in *ClassB*
  - there is an import statement in *ClassA* for *ClassB*
  - there is an import statement in *ClassB* for *ClassA*



UML DIAGRAMS

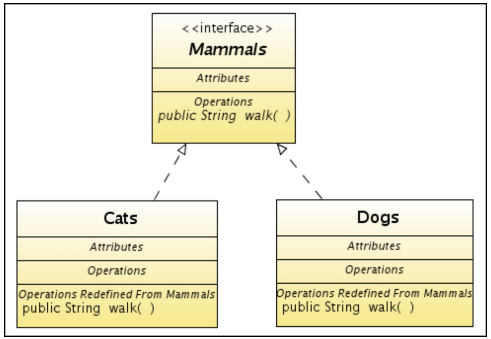
- Dependency relationship
  - a generalized connection between two classes if other relationships are not meaningful!
  - We will primarily use it for classes referenced in **main()**.
- *ClassA* depends on *ClassB* implies one or more of the following
  - at least one *ClassB* object is referenced in *ClassA*
  - there is an import statement in *ClassA* for *ClassB*
  - at least one class method in *ClassB* is called by *ClassA*

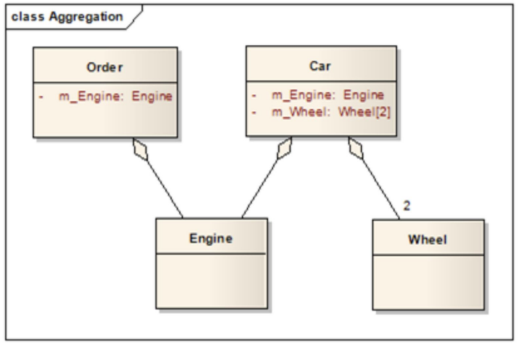
UML DIAGRAMS

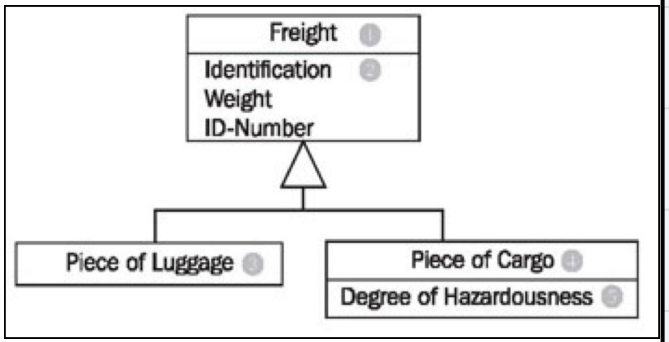
- Dependencies and relationships between classes can be
  - General Dependency relationship *ClassA* ⇢ *ClassB*
  - Unidirectional association *ClassA* → *ClassB*
  - Bidirectional association *ClassA* — *ClassB*
  - Aggregation relationship *ClassA* ◇— *ClassB*
  - Composition relationship *ClassA* ◆— *ClassB*
  - Realization relationship *ClassA* ⇢▷ *ClassB*
  - Generalization relationship *ClassA* —▷ *ClassB*

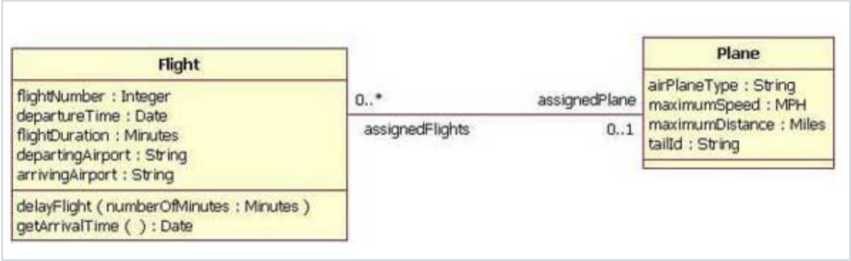
UML DIAGRAMS

- You might see UML diagrams in different formats









UML RESOURCES

- UML relationships & associations
- UML.org
- The UML 2 class diagram - IBM Developer

THANK YOU!

DO YOU HAVE ANY QUESTIONS?

hend.alkittawi@utsa.edu

By Appointment

Online