Lecture 3 Outline

• Object and Class
• How to Define Class
• A Glimpse on Designing a Class
• Object Encapsulation
Class and Object
Class and Object

Class

- Class - A blueprint of an object, does not occupy physical space (memory) during run-time except its static members
Class and Object

Class
• Class - A blueprint of an object, does not occupy physical space (memory) during run-time except its static members

Object instance 1
• Object instance - A concrete entity that resides inside the memory during run-time with its own field contents
Class and Object

- **Class** - A blueprint of an object, does not occupy physical space (memory) during run-time except its static members.

- **Object instance** - A concrete entity that resides inside the memory during run-time with its own field contents.

- Object instances share same field data structure, but not necessarily the same field status.
Define Class - Member Field

- Variables declared directly inside a class

```java
class Dog {
    static final String default_breed = "German Shepherd";
    static final String default_name  = "Tuesday";

    String name;
    int age;
    String breed;
}
```
Define Class - Member Field

- Variables declared directly inside a class
- Member field lives in the entire scope of the class definition
Define Class - Member Field

- Variables declared directly inside a class
- Member field lives in the entire scope of the class definition
- The values of the instance(non-static) member fields determines the status of an object (memory allocation per object instance)
Define Class - Member Field

- Variables declared directly inside a class
- Member field lives in the entire scope of the class definition
- The values of the instance(non-static) member fields determines the status of an object (memory allocation per object instance)
- The values of the static member fields are uniform across all object instances (memory allocation per class)
Define Class - Constructor, a Special Member Method

class Dog {
    static String default_breed = "German Shepherd";
    static String default_name = "Tuesday";
    String name;
    int age;
    String breed;

    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
}
Define Class - Constructor, a Special Member Method

```java
class Dog {
    static String default_breed = "German Shepherd";
    static String default_name  = "Tuesday";
    String name;
    int age;
    String breed;

    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
}
```

- Special that it is dedicated for holding what to do during the instantiation of this class.
Define Class - Constructor, a Special Member Method

```java
class Dog {
    static String default_breed = "German Shepherd";
    static String default_name = "Tuesday";
    String name;
    int age;
    String breed;

    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
}
```

- A constructor without parameters

- Special that it is dedicated for holding what to do during the instantiation of this class.

```java
Dog rebel = new Dog();
```
Define Class - Constructor, a Special Member Method

class Dog {
    static String default_breed = "German Shepherd";
    static String default_name  = "Tuesday";
    String name;
    int age;
    String breed;

    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
}

A constructor without parameters

• Special that it is dedicated for holding what to do during the instantiation of this class.

• Thus, must use the class name as its name

Dog rebel = new Dog();
Define Class - Constructor, a Special Member Method

- A constructor without parameters
  - Special that it is dedicated for holding what to do during the instantiation of this class.
  - Thus, must use the class name as its name
  - Thus, NO return type, NO return statement
Define Class - Member Method

- Describe the behavior of this type of object

```java
class Dog {
    ...
    String name;
    int age;
    String breed;
    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
    public String toString() {
        String result = "My name is " + name
                          + ", and I am a " + age
                          + " years old " + breed + " dog."
                          + "
        return result;
    }
    void changeName(String name) {
        this.name = name;
    }
    void grow() {
        ++age;
    }
}
```
Define Class - Member Method

- Describe the behavior of this type of object

- **Accessor** method - return object status to clients, **getter**

```java
class Dog {
    ... 
    String name;
    int age;
    String breed;
    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
    public String toString() {
        String result = "My name is " + name 
                        + ", and I am a " + age 
                        + " years old " + breed + " dog."
        return result;
    }
    void changeName(String name) {
        this.name = name;
    }
    void grow() {
        ++age;
    }
}
```
Define Class - Member Method

- Describe the behavior of this type of object
- **Accessor** method - return object status to clients, **getter**

```java
class Dog {
    String name;
    int age;
    String breed;
    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
    public String toString() {
        String result = "My name is " + name + ", and I am a " + age + " years old " + breed + " dog."
        return result;
    }
    void changeName(String name) {
        this.name = name;
    }
    void grow() {
        ++age;
    }
    Dog puppy = new Dog();
    System.out.println(puppy);
}
```

Sunday, January 30, 2011
Define Class - Member Method

- Describe the behavior of this type of object
- **Accessor** method - return object status to clients, **getter**
- **Mutator** method - modify object status according to the parameters passed from clients, **setter**

```java
class Dog {
    ... 
    String name;
    int age;
    String breed;
    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
    public String toString() {
        String result = "My name is " + name 
                        + ", and I am a " + age 
                        + " years old " + breed + " dog."
        return result;
    }
    void changeName(String name) {
        this.name = name;
    }
    void grow() {
        ++age;
    }
}
```

Dog puppy = new Dog();
System.out.println(puppy);

Sunday, January 30, 2011
Define Class - Member Method

- Describe the behavior of this type of object
- **Accessor** method - return object status to clients, **getter**
- **Mutator** method - modify object status according to the parameters passed from clients, **setter**
Define Class - Scope

- A declared item lives only within the code block where the declaration stands.

class Dog {
    ...
    String name;
    int age;
    String breed;
    Dog(String aName){
        name = aName;
        age = 0;
        breed = default_breed;
    }
    void grow(){
        ++age;
    }
}
Define Class - Scope

• A declared item lives only within the code block where the declaration stands.

• The scope of a declared item refers to its residing code space.

```java
class Dog {
    String name;
    int age;
    String breed;

    Dog(String aName) {
        name = aName;
        age = 0;
        breed = default_breed;
    }

grow() {
    ++age;
}
}
```
Define Class - Scope

• A declared item lives only within the code block where the declaration stands.

• The scope of a declared item refers to its residing code space.

• The scope sheds through all the sub-blocks
Define Class - Scope

- A declared item lives only within the code block where the declaration stands.
- The scope of a declared item refers to its residing code space.
- The scope sheds through all the sub-blocks

```java
class Dog {
    ...
    String name;
    int age;
    String breed;
    Dog(String aName) {
        name = aName;
        age = 0;
        breed = default_breed;
    }
    void setAge(int age) {
        age = age;
    }
    void grow() {
        ++age;
    }
}
```
Define Class - Scope

• A declared item lives only within the code block where the declaration stands.

• The scope of a declared item refers to its residing code space.

• The scope sheds through all the sub-blocks

```java
class Dog {
    ...
    String name;
    int age;
    String breed;
    Dog(String aName) {
        name = aName;
        age = 0;
        breed = default_breed;
    }

    void grow() {
        ++age;
    }
}
void setAge(int age) {
    age = age;
}
```
Local var shadows Instance Member
Define Class - Scope

• A declared item lives only within the code block where the declaration stands.

• The scope of a declared item refers to its residing code space.

• The scope sheds through all the sub-blocks

• Use this reference to refer to the receiver object (this. -> this receiver object instance’s)
Define Class - Overloading

class Dog {
...
	Dog(){
		name = default_name;
		age = 0;
		breed = default_breed;
	}

	Dog(String name, int age){
		this.name = name;
		this.age = age;
		this.breed = default_breed;
	}

	Dog(String name, int age, String breed){
		this.name = name;
		this.age = age;
		this.breed = breed;
	}
...
}
Define Class - Overloading

- Member methods with the same identifier, but different parameter lists (method signature)

```java
class Dog {
    ... 
    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }
    Dog(String name, int age) {
        this.name = name;
        this.age = age;
        this.breed = default_breed;
    }
    Dog(String name, int age, String breed) {
        this.name = name;
        this.age = age;
        this.breed = breed;
    }
    ...
}
```
Define Class - Overloading

- Member methods with the same identifier, but different parameter lists (method signature)

```java
class Dog {
    Dog() {
        name = default_name;
        age = 0;
        breed = default_breed;
    }

    Dog(String name, int age){
        this.name = name;
        this.age = age;
        this.breed = default_breed;
    }

    Dog(String name, int age, String breed){
        this.name = name;
        this.age = age;
        this.breed = breed;
    }
    ...
}
```
Define Class - Overloading

- Member methods with the same identifier, but different parameter lists (method signature)

- Which method to be invoked depends on the actual parameter list
Define Class - Overloading

- Member methods with the same identifier, but different parameter lists (method signature)
- Which method to be invoked depends on the actual parameter list

Dog puppy = new Dog();
Dog daisy = new Dog("Daisy", 4);
Dog denny = new Dog("Denny", 10, "Golden Retriver");
A Glimpse on Designing Class
-When to create a class

A student management system that tracks college students’ information including the name, student id, contact info, their earned academic credit, etc.
A Glimpse on Designing Class
-When to create a class

A student management system that tracks college
students’ information including the name,
student id, contact info, their earned academic
credit, etc.
A Glimpse on Designing Class
-When to create a class

A student management system that tracks college students’ information including the name, student id, contact info, their earned academic credit, etc.
A Glimpse on Designing Class
- When to create a class

A student management system that tracks college students’ information including the name, student id, contact info, their earned academic credit, etc.
A Glimpse on Designing Class

- When to create a class

A student management system that tracks college students’ information including the name, student id, contact info, their earned academic credit, etc.

- When you see a pattern of data structure and its corresponding manipulations, and it might appear in the application for many times.
A Glimpse on Designing Class
- When to create a class

• When you see a pattern of data structure and its corresponding manipulations, and it might appear in the application for many times.

• Class - Abstraction of common data and methods for a type of objects

A student management system that tracks college students’ information including the name, student id, contact info, their earned academic credit, etc.
What Goes Into a class?

Student

- String name;
- int id;
- String contact;
- int credit_earned;

Student(String name, int id);
String toString();
void setContact(String contact);
String getContact();
void initCredit();
void addCredit(int credit);

boolean ready2graduate();

boolean validateCredit();
boolean collectTransferCredit();

data

methods
What Goes Into a class?

- Member data

```java
String name;
int id;
String contact;
int credit_earned;

Student(String name, int id);
String toString();
void setContact(String contact);
String getContact();
void initCredit();
void addCredit(int credit);
boolean ready2graduate();

boolean validateCredit();
boolean collectTransferCredit();
```
What Goes Into a class?

- Member data
- Internal items that compose the object

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>String name;</td>
</tr>
<tr>
<td>int id;</td>
</tr>
<tr>
<td>String contact;</td>
</tr>
<tr>
<td>int credit_earned;</td>
</tr>
</tbody>
</table>

| Student(String name, int id); |
| String toString(); |
| void setContact(String contact); |
| String getContact(); |
| void initCredit(); |
| void addCredit(int credit); |
| boolean ready2graduate(); |
| boolean validateCredit(); |
| boolean collectTransferCredit(); |
What Goes Into a class?

- Member data
  - Internal items that compose the object
- Member method

```
Student
String name;
int id;
String contact;
int credit_earned;

Student(String name, int id);
String toString();
void setContact(String contact);
String getContact();
void initCredit();
void addCredit(int credit);

boolean ready2graduate();
boolean validateCredit();
boolean collectTransferCredit();
```
What Goes Into a class?

- Member data
  - Internal items that compose the object
- Member method
  - Associated data manipulations that serve whoever outside of the class (interface methods)

```
Student
String name;
int id;
String contact;
int credit_earned;

Student(String name, int id);
String toString();
void setContact(String contact);
String getContact();
void initCredit();
void addCredit(int credit);
boolean ready2graduate();

boolean validateCredit();
boolean collectTransferCredit();
```
What Goes Into a class?

- **Member data**

  - Internal items that compose the object

- **Member method**

  - Associated data manipulations that serve whoever outside of the class (interface methods)

  - Associated data manipulations that are not directly used by the outsiders

---

**Student**

<table>
<thead>
<tr>
<th>data</th>
<th>methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>String name; int id; String contact; int credit_earned;</td>
<td>Student(String name, int id); String toString(); void setContact(String contact); String getContact(); void initCredit(); void addCredit(int credit); boolean ready2graduate(); boolean validateCredit(); boolean collectTransferCredit();</td>
</tr>
</tbody>
</table>
What Goes Into a class?

- Member data
  - Internal items that compose the object
- Member method
  - Associated data manipulations that serve whoever outside of the class (interface methods)
  - Associated data manipulations that are not directly used by the outsiders
- The choice of modifiers to each member:

```java
Student(String name, int id);
String toString();
void setContact(String contact);
String getContact();
void initCredit();
void addCredit(int credit);
boolean ready2graduate();
boolean validateCredit();
boolean collectTransferCredit();
```
What Goes Into a class?

- Member data
  - Internal items that compose the object

- Member method
  - Associated data manipulations that serve whoever outside of the class (interface methods)
  - Associated data manipulations that are not directly used by the outsiders

- The choice of modifiers to each member:
  - final, static, (public, private), etc
Instance or Static Member

• Which member needs to be static? A few typical scenario:
Instance or Static Member

- Which member needs to be static? A few typical scenario:

  - If this class does not hold any instance member field (e.g. Math class)
Instance or Static Member

- Which member needs to be static? A few typical scenario:
  - If this class does not hold any instance member field (e.g. Math class)
  - If a member is a constant
Instance or Static Member

• Which member needs to be static? A few typical scenario:

  • If this class does not hold any instance member field (e.g. Math class)

  • If a member is a constant

  • If that member is general to all instances of this class (e.g. an instance counter in a class)
Instance or Static Member

- Which member needs to be static? A few typical scenarios:
  
  - If this class does not hold any instance member field (e.g. Math class)
  
  - If a member is a constant
  
  - If that member is general to all instances of this class (e.g. an instance counter in a class)

```java
class Counter{
    static int numInstances = 0;
    Counter(){ ++numInstances;}
}
```
Instance or Static Member

• Which member needs to be static? A few typical scenario:

  • If this class does not hold any instance member field (e.g. Math class)

  • If a member is a constant

  • If that member is general to all instances of this class (e.g. an instance counter in a class)

```java
class Counter{
    static int numInstances = 0;
    Counter(){ ++numInstances; }
}
```
Instance or Static Member

- Which member needs to be static? A few typical scenario:
  - If this class does not hold any instance member field (e.g. Math class)
  - If a member is a constant
  - If that member is general to all instances of this class (e.g. an instance counter in a class)

```java
class Counter{
    static int numInstances = 0;
    Counter(){ ++numInstances;}
}
```

- Special designs: Singleton design pattern (only one instance is allowed to be generated)
Object Encapsulation - Public or Private Member

• Class member (field or method) visibility modifier

  • **public**: sets the member visible to any classes

  • **private**: sets the member visible only to this class

  • **protected**: (Will be covered in Chapter 8 Inheritance)

  • default visibility is “public to the classes within the current package”

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>- String name;</td>
</tr>
<tr>
<td>- int id;</td>
</tr>
<tr>
<td>- String contact;</td>
</tr>
<tr>
<td>- int credit_earned;</td>
</tr>
<tr>
<td>+ Student(String name, int id);</td>
</tr>
<tr>
<td>+ String toString();</td>
</tr>
<tr>
<td>+ void setContact(String contact);</td>
</tr>
<tr>
<td>+ String getContact();</td>
</tr>
<tr>
<td>+ void initCredit();</td>
</tr>
<tr>
<td>+ void addCredit(int credit);</td>
</tr>
<tr>
<td>+ boolean ready2graduate();</td>
</tr>
<tr>
<td>- boolean validateCredit();</td>
</tr>
<tr>
<td>- boolean collectTransferCredit();</td>
</tr>
</tbody>
</table>
Object Encapsulation - Public or Private Member

- Class member (field or method) visibility modifier

- public: sets the member visible to any classes

- private: sets the member visible only to this class

- protected: (Will be covered in Chapter 8 Inheritance)

- default visibility is “public to the classes within the current package”

---

Student

- String name;
- int id;
- String contact;
- int credit_earned;

- Student(String name, int id);
- String toString();
- void setContact(String contact);
- String getContact();
- void initCredit();
- void addCredit(int credit);
- boolean ready2graduate();
- boolean validateCredit();
- boolean collectTransferCredit();
Object Encapsulation - Public or Private Member

- Class member (field or method) visibility modifier

  - **public**: sets the member visible to any classes

  - **private**: sets the member visible only to this class

  - **protected**: (Will be covered in Chapter 8 Inheritance)

- default visibility is “public to the classes within the current package”
Public or Private Member - Object Encapsulation - continued

- Hide instance member fields, so their status can only be manipulated through the corresponding methods

- Internal changes in class A does not affect its clients as long as class A has stabled public method interface

- Information hiding helps maintaining consistent dependency among classes
Demo and Summary

- Class - A pack of members in data (or fields) and/or methods
- Constructor - The method to be called by the new operator
- Method overloading - same identifier, different parameter list
- Modifiers on class members: static; public, private
- Object encapsulation through the distribution of visibility and functionalities among class members
Lecture 4 Outline

- Parameter passing - Primitive vs Class types
- Class Relationship 1 - Dependency
- Class Relationship 1.1 - Aggregation
Parameter Passing - Primitive vs Class types

- Every parameter is passed by value in Java.

- Objects are never passed in method calls. It is their references that are passed.

- For a class-type parameter, if any mutator method is been called inside the callee method, the changed status is visible to the caller method.

- Primitive-type parameters always stay the same before and after a method call.
Class Relationships - Dependency

- Dependency: one class relies on the implementation of another class
- A depends on B - “Use a” relationship
- One class can pass parameters of object references in this class “Self-Use”
Class Relationships - Aggregation

- A class has member fields in another class
- Class A has a Class B object - the “has-a” relationship
- A special case Dependency relationship
Class Relationships - Aggregation

- A class has member fields in another class
- Class A has a Class B object - the "has-a" relationship
- A special case Dependency relationship
Summary

- Object is never passed, Object reference is passed
- Often we will be creating more than just one class in the future
- One class should only implement the concept of one noun (or object)
- When one class’s internal data structure is not cleanly organized, make new classes to group data for a sub-object, and use the aggregation relationship to put the classes together.