### Previous Lecture:
*Object oriented programming*
- Objects and classes
- Methods
- Modifiers

### Today’s Lecture:
- Review (objects, classes, methods)
- Method implementation and use
- Constructor

### Reading (JV):
- Sec 4.3-4.5, 5.1, 5.2

---

### Review questions

- What is the difference between instance and class variables (and methods)?

- How do you recognize a class variable (or method)?

  Declaration syntax:
  
  ```
  modifiers type name;
  ```

  Examples from Project 5:
  ```
  private int id;
  private static int nextID;
  Room now;
  ```

### Methods

**Syntax:**

```java
return-type method-name ( parameter-list ) {
  statement-list
}
```

- Modifiers can be used in the header
- `return-type` can be any primitive or non-primitive types
- `return-type void` means nothing is returned from the method
- Methods that return values must have a return statement:
  ```
  return expression;
  ```
- Static methods cannot reference instance variables
Example class definition

```java
public class Account {
    private int balance; // current bal.
    private int deposits; // deposits to date
    private int withdrawals; //withdrawal to date

    // deposit d to account
    public void deposit(int d) {
        balance = balance + d;
        deposits = deposits + d;
    }

    // withdraw w from account
    public void withdraw(int w) {
        balance -= w;
        withdrawals += w;
    }
}
```

“Calling” a method

- Class (static) method:
  ```java
class-name.method-name(expression-list)
  ```
- Instance method:
  ```java
reference-variable.method-name(expr-list)
  ```

Using instance methods

- Let \( m \) be a method in objects of class \( c \)
- Let \( r \) refer to object \( o \) of class \( c \)
- Then expression
  ```java
  r.m(expression-list)
  ```
invokes method \( m \) in object \( o \)
- When method \( m \) is executed, field names signify the instance variables in object \( o \)
- Example: \( account1.deposit(200); \)

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>200</td>
</tr>
<tr>
<td>deposits</td>
<td>200</td>
</tr>
<tr>
<td>withdrawals</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Constructor

- A **constructor** is used to create objects
- Each class has a default constructor
- Can define your own constructor:
  ```java
  modifiers class-name( parameter-list ) {
  statements-list
  }
  ```
- Use `public` as the modifier for now
- Note: Looks like a method but it has **no** return type
Constructor invocation

```
new class-name(expression-list)
```

- The value of above expression is a reference to a new object of the given class-name
- The defined (or default) constructor is invoked on the new object created by `new`

Example constructor

```java
public class Room {
    private static int nextID = 1;
    private int id;

    // Constructor
    public Room() {
        id = nextID;
        nextID = nextID + 1;
    }
}
```

After 3 calls of `new Room()` ...

- `nextID`
- `id`
- `id`
- `id`

Class Room

- Class `Room` is a collection of rooms
- Each `Room` has a unique ID#
- Method `findRoom(int targetID)` returns the `Room` with ID# equal to `targetID` or returns null if there is no such room
- But how to find the `Room`?
- Idea: keep all `Rooms` chained together
Three instances of Room

nextID: 4

id: 1
previous: null

id: 2
previous: 1

id: 3
previous: 2

last: 3

Class definition

public class Room {

    // Each Room r has unique r.id>0.
    // nextID is ID# of next Room to be
    // created.
    private static int nextID;
    private int id;

    // In each Room r, r.previous is the
    // Room created immediately before r,
    // or isnull if r is 1st Room.
    // last is the most recent Room to
    // have been created.
    private static Room last;
    private Room previous;

    // Constructor to create a chain of
    // Rooms
    public Room() {
        id = nextID;
        nextID = nextID + 1;
        previous = last;
        last = this;
    }
}

Constructor to create a chain of Rooms

public class Room {

    private static int nextID;
    private int id;
    private static Room last;
    private Room previous;

    // Constructor to create a chain of
    // Rooms
    public Room() {
        id = nextID;
        nextID = nextID + 1;
        previous = last;
        last = this;
    }
}