Object-Oriented Programming

Classes

Object is an instance of a house. Contains bdrs (number of bedrooms) and baths (number of bathrooms).

Methods in object refer to fields in object. Could have an array of such objects to list the apartments in a building.

With variables h1 and h2 below,

t1.getBeds() is 2

t2.getBeds() is 4

Class invariants

/** An instance maintains info for a house */

public class House {

private int bdrs; // number of bedrooms, in 0…10

private int baths; // number of bathrooms, in 1…5

Software engineering principle: Always write a clear, precise class invariant, which describes all fields.

Call of every method starts with class invariant true and should end with class invariant true.

Frequent reference to class invariant while programming can prevent mistakes.

}``

Generate javadoc

- With project selected in Package explorer, use menu item Project -> Generate javadoc.
- In Package Explorer, click on the project -> doc -> index.html
- You get a pane with an API like specification of class Time, in which javadoc comments (start with /**) have been extracted!

- That is how the API specs were created.

Class House

/** An instance maintains info for a House */

public class House {

private int bdrs; // number of bedrooms, in 0…10

private int baths; // number of bathrooms, in 1…5

Access modifier private:
can’t see field from outside class

Software engineering principle: make fields private, unless there is a real reason to make public

}``
**Getter methods (functions)**

```java
/** An instance maintains info for a house */
public class House {
    private int bdrs; // number of bedrooms, in 0..10
    private int baths; // number of bathrooms, in 1..5

    /** Return number of bedrooms */
    public int getBeds() {
        return bdrs;
    }

    /** Return number of bathrooms */
    public int getBaths() {
        return baths;
    }
}
```

**Setter methods (procedures)**

```java
/** An instance maintains info for a house */
public class House {
    private int bdrs; // number of bedrooms
    private int baths; // number of bathrooms,
    ...    /** Change number of bathrooms to b */
    public void setBeds(int b) {
        bdrs = b;
    }
}
```

**Method specs should not mention fields**

```java
/**
 * return number of rooms
 */
private int getRooms() {
    return bdrs + baths;
}
```

**A little about type {class} String**

```java
/**
 * Return a representation of this house *
 */
private String toString() {
    return plural(bdrs) + " bedrooms, 
    plural(baths) + " bathrooms."
}
```

**Test using a JUnit testing class**

```java
public class HouseTester {
    @Test
    public void testSetters() {
        House h = new House();
        h.setBeds(2);
        assertEquals(2, h.getBeds());
    }
}
```
Test setter method in JUnit testing class

```java
public class HouseTester {
    …
    @Test
    public void testSetters() {
        House h = new House();
        h.setBeds(2);
        assertEquals(2, h.getBeds());
    }
}
```

HouseTester can have several test methods, each preceded by `@Test`. All are called when menu item Run → Run is selected.

Constructors — new kind of method

```java
public class C {
    private int a;
    private int b;
    private int c;
    private int d;
    private int e;
}
```

C has lots of fields. Initializing an object can be a pain — assuming there are suitable setter methods.

```java
C var = new C();
var.setA(2);
var.setB(20);
var.setC(35);
var.setD(-15);
var.setE(150);
```

But first, must write a new method called a constructor.

```java
public House(int bd, int bth) {
    bdrs = bd;
    baths = bth;
}
```

Purpose of constructor: Initialize fields of a new object so that its class invariant is true.

```java
public House() {
    bdrs = 0;
    baths = 0;
}
```

Revisit the new-expression

```java
public class House {
    private int bdrs; // number of bedrooms, in 0..10
    private int baths; // number of bathrooms, in 1..5
}
```

```java
/** An object maintains info about a house. */
public class House {
    private int bdrs; // number of bedrooms, in 0..10
    private int baths; // number of bathrooms, in 1..5
    /** Constructor: an instance with bd bedrooms and bth bathrooms. 
     * Precondition: bd in 0..10, bth in 1..5. 
     */
    public House(int bd, int bth) {
        bdrs = bd;
        baths = bth;
    }
    …
}
```

```java
Evaluation of new-expression:
1. Create a new object of class <constructor-call>, with default values in fields
2. Execute the constructor-call
3. Give as value of the expression the name of the new object

If you do not declare a constructor, Java puts in this one:
```java
public <class-name>() { }
```

Recap

- An object is defined by a class. An object can contain variables (fields) as well as methods (functions/procedures).
- Use comments and javadoc to document invariants and specify behavior.
- Generally, make fields private so they can't be seen from outside the class. May add getter methods (functions) and setter methods (procedures) to allow access to some or all fields.
- Use a new kind of method, the constructor, to initialize fields of a new object during evaluation of a new-expression.
- Create a JUnit Testing Class to save a suite of test cases.
**CS2110 FAQs**

- **Lecture Videos:** they're available [http://cornell.videonote.com/channels/1027/videos](http://cornell.videonote.com/channels/1027/videos)
- **Grading Options:** S/U is fine by us. Check with your advisor/major.
- **Prelim conflicts:** Please don’t email us about prelim conflicts! We’ll tell you at the appropriate time how we handle them.
- **Other Questions:** check course Piazza regularly for announcements.

**Recitation This Week**

You must read/watch the tutorial BEFORE the recitation: [www.cs.cornell.edu/courses/cs2110/2017fa/online/exceptions/EX1.html](http://www.cs.cornell.edu/courses/cs2110/2017fa/online/exceptions/EX1.html)

Get to it from the Tutorials page of the course website. **NOTE THAT THERE ARE SIX WEB PAGES!**

Bring your laptop to class, ready to answer questions, solve problems. During the section, you can talk to neighbors, discuss things, answer questions together. The TA will walk around and help. The TA will give a short presentation on some issue if needed.

Homework questions are on the course website. You will have until a week after the recitation (on a Wednesday night) to submit answers on the CMS.

**Assignments**

- **A0 out:** Due this Thursday (8/31)
- **A1 out:** Due next Wednesday (9/6)
- **A2 out:** Due the following week (9/13)

**Assignment A1**

Write a class to maintain information about PhDs ---e.g. their advisor(s) and date of PhD.

Objectives in brief:

- Get used to Eclipse and writing a simple Java class
- Learn conventions for Javadoc specs, formatting code (e.g. indentation), class invariants, method preconditions
- Learn about and use JUnit testing

Important: READ CAREFULLY, including Step 7, which reviews what the assignment is graded on.

**Assignments**

Groups. You can do A1 with 1 other person. FORM YOUR GROUP EARLY! Use Piazza Note @5 to search for partner!

CHECK the pinned A1 note on the Piazza every day.