Public, Static and Arrays

CS61BL Summer 2009
6/25/2009

Access Levels (HFJ pg 667)

- **public** – any code anywhere can access the public thing
- **protected** – same as default + subclasses outside the package can inherit the protected thing.
- **default** – only code within the same package as the class with the default thing can access the default thing
- **private** – only code within the same class can access the private thing.

By “thing” we mean class, variable, method, constructor

Public vs. Private Variables

- **public** variables are accessible outside of the class
- **private** variables are only accessible inside of the class
- When we get to inheritance - if you have a sub-class, it can’t access the **private** variables

Public vs. Private Methods

- Private methods are not accessible by other classes
- Helper methods should be private

static fields

- Global state shared by all instances of the class
- Looks like an instance variable declaration + static
- Access them as ClassName.fieldName

    public class Person {
        int myAge;
        String myName;
        Person myParent;
        static int numberOfPeople = 0;
        public Person(int age, String name) {
            this.myAge = age;
            this.myName = name;
            Person.numberOfPeople ++;
        }
    }

static methods

    public class Person {
        int myAge;
        String myName;
        Person myParent;
        static int numberOfPeople = 0;
        public static int getNumPeople() {
            return Person.numberOfPeople;
        }
        public static void main(String[] args) {
            Person.getNumPeople();
        }
    }
**static methods**

- static methods do not operate on an object
- Has no implicit parameter this
- Can NOT access instance variables
- Can access static fields
- Call the method as `ClassName.methodName(...)`

```java
static version
public double sqrtAge() {
    return Math.sqrt(myAge);
}

Non static version
public double sqrtAge2() {
    Math myMathObj = new Math();
    return myMathObj.sqrt(myAge);
}
```

---

**Announcements**

- Join the google group (link from the webpage)
- Don’t worry about your quiz scores!!!
- Do the reading
- PRACTICE, PRACTICE, PRACTICE!

---

**Debugging an Array**

*After the first line:*

*After the second line:*

---

**Debugging an Array**

*After the first line:*

*After the second line:*
Arrays

```java
int [] notes = {64, 62, 62, 62, 64, 64, 64, 64, 64, 64, 64, 62};
System.out.println(notes[notes.length-1]);
```

For Loops

```java
for (int i = 0; i < 10; i++)
{
    System.out.println(i + " ");
}
```

Awesome For Loops (FOR EACH)

```java
int [] notes = {60, 61, 62};
for (int oneNote : notes)
{
    System.out.println(oneNote + " ");
}
```
Arrays

- Set to the default value for primitive types
- Set to null for objects
  
  ```java
  Type [] arrayName = new Type[size];
  Type [] arrayName = {value1, value2};
  ```

- Arrays don’t change size
- Arrays hold one type of thing
- Access items in the array using the [] notation
  ```java
  arrayName[pos] = newValue;
  ```

Flow of control

- break – breaks from the inner most loop enclosing the break and continues executing at the code after the loop.
- return – causes the method to end immediately and return to the calling method
- continue – transfers control to the header of the inner most enclosing loop.