Announcements

- Help sessions:
  - Every Wednesday 4:30 -- 6:30
  - BRNG 2280
  - Except,
    - **not** this Wednesday, but **today**
      - 5:00 - 6:30 in SMTH 108
  - November 28th
    - ARMS 1010

Monday, October 15, 12

User Defined Classes
Part 2

CS 180
Sunil Prabhakar
Department of Computer Science
Purdue University

Class vs. Instance methods

- Compare the Math and String class methods that we have used:
  - Math.pow(2,3);
  - str.charAt(4);
- One is called using the class name (Math) and the other using an object identifier (str)
- How are they different?
- pow() is a *class method* whereas charAt is an *instance method*

Monday, October 15, 12

Class Methods

- A *class method* is one which is defined with the **static** keyword.

- A class (**static**) method
  - can be called directly from any other static method of the class
  - can be called from anywhere within the same class; or using the class name from another class (if it is public)
  - cannot directly call an instance method of the same class -- must be called on an object of that class
  - cannot access instance data members -- only through an object

Monday, October 15, 12

Instance Methods

- An *instance (object) method* is one which is defined without the static keyword.
- An instance method
  - can be called directly from any other instance method of the class
  - can be called on an object of that class from either a static method or a method of another class.
  - can call a class method of the same class
  - can call another object method of the same class without using an object

Monday, October 15, 12

Class and Instance Methods

```java
public class Test{

  int id;

  public void methodA() {
    id = 5;
    methodB();
  }
  public void methodB() {
    id = 7;
    methodT();
    Test.methodT();
  }
  public static void methodS() {
    Test t = new Test();
    methodT();
    t.methodA();
    methodA();
    id = 3;
  }
  public static void methodT() {
    . . .
  }
}
```

instance method can call a class method without the class name.

class method can't call instance method without an object.

class method can't access an instance data member without an object.

© Sunil Prabhakar, Purdue University

PURDUE
UNIVERSITY

Monday, October 15, 12

Calling Methods of Same Class

```java
class Student {
    private String id;
    private String name;

    public void setName(String newName){
        . . .
    }
    private void setId(String newID){
        setName("ID changed");
        . . .
    }
    public String getId(){
        . . .
    }
```

No object specified.

setName() will use the same object as the one on which setId() was called.

PURDUE
UNIVERSITY

7

Monday, October 15, 12

Class Data Members

- As with methods, we can have both instance data members and class data members.

- Both are defined outside any method.

- Class data members are those with the static keyword.

- There is only one copy of each class data member -- shared by all objects of the class

- Each object has its own copy of all instance data members.
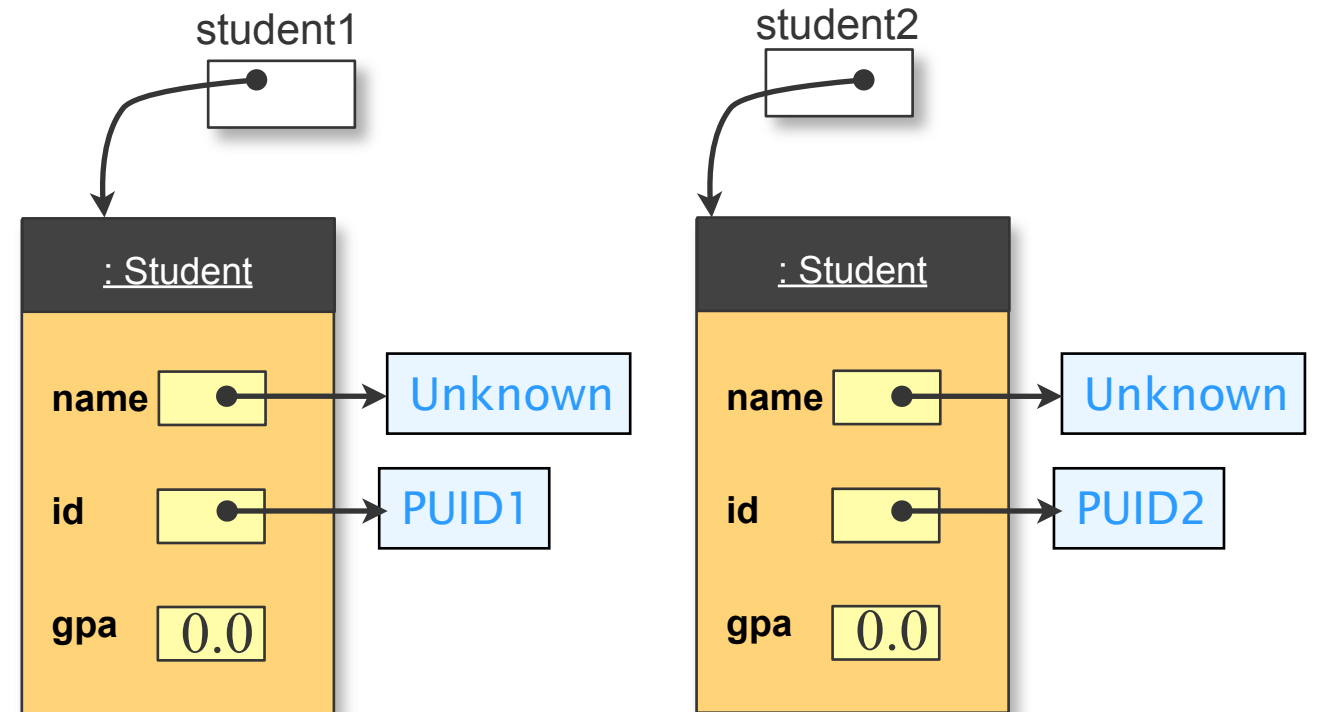
Monday, October 15, 12

Problem

- *Consider the Student class discussed earlier.*
- *Modify it so that student IDs are automatically generated as follows:*
  - *each id is of the form: PUIDnnnn where nnnn is a number.*
  - *each successive student object created should get a unique id is sequence, starting with PUID1, then PUID2, PUID3, ...*

Monday, October 15, 12

Solution

- How will the constructor know what was the last ID generated for a student?
- We have to save this somewhere.
  - Where?
  - This can't be stored as a data member. Why?
- This information can be saved in a class data member.
  - There is only one copy of it for the entire class.

PURDUE
UNIVERSITY

Monday, October 15, 12

Student

```java
class Student {
  static int nextID = 1;

  private String name, id;
  private double gpa;

  public Student(){
    id = "PUID" + Student.nextID;
    Student.nextID++;
    name = "Unknown";
    gpa = 0.0;
  }
    . . .
}
```

student1

: Student

name	●	→	Unknown
id	●	→	PUID1
gpa	0.0		

student2

: Student

name	●	→	Unknown
id	●	→	PUID2
gpa	0.0		

```java
Student student1, student2;

student1 = new Student();

student2 = new Student();
```

Student

| nextID | 𝔅 |

NOTE: Class, **not** object.

PURDUE
UNIVERSITY

Monday, October 15, 12
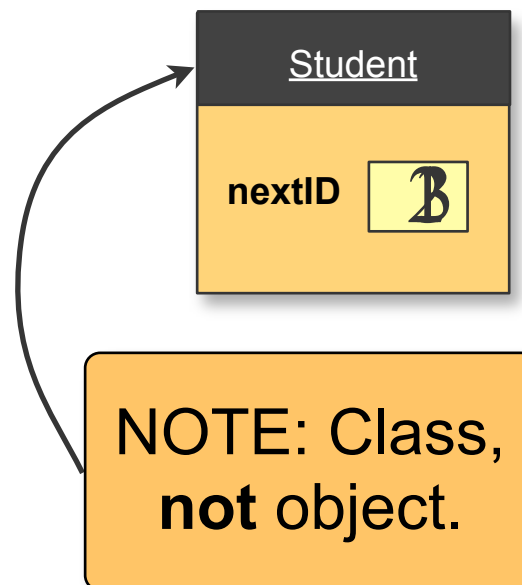
Class Data Members

- Class data members can be
  - directly accessed by class or instance methods of the same class
  - can be constants (defined with **static final**)
  - instance constants should usually be class constants (less space is used).

PURDUE
UNIVERSITY

Monday, October 15, 12

Main method

- Recall that main is a class method.
- Thus, it can't call any instance methods directly.
  - we must have an object to do this
- Any methods called from main should also be class methods.
- None of these class methods can access instance data members.

PURDUE
UNIVERSITY

Monday, October 15, 12

Recall: Roster Class

```java
public class Roster{
  public static void main(String[] args) {
    Student[] studentList;
    Student student1;

    studentList = initializeRoster();
    student1 = findStudent("2334", studentList);
    if(student1 == null)
        System.out.println("Student with id 2334 not found in class");
     else
        student1.printNeatly();
  }
  public static Student findStudent(String id, Student[] sList){
    Student s;
    int i;
    for(i=0; i < sList.length;i++)
       if(id.equals(sList[i].getId()))
          return sList[i];
     return null;
  }
}
```

studentList is a local variable in main.

Need to pass studentLIst as a parameter.

PURDUE
UNIVERSITY

14

Monday, October 15, 12

Alternative:Class Data Member

```java
public class RosterV2{
  static Student[] studentList;

  public static void main(String[] args) {
    Student student1;

    studentList = initializeRoster();
    student1 = findStudent("2334");
    if(student1 == null)
        System.out.println("Student with id 2334 not found in class");
    else
        student1.printNeatly();
  }
  public static Student findStudent(String id){
    int i;
    for(i=0; i < studentList.length;i++)
        if(id.equals(studentList[i].getId()))
            return studentList[i];
    return null;
  }
}
```

studentList is a class data member.

No need to pass studentLIst as a parameter.

studentLIst accessible directly by a class method.

© Sunil Prabhakar, Purdue University

PURDUE
UNIVERSITY

Monday, October 15, 12

Alternative: Instance Data Member

```java
public class RosterV3{
  Student[] studentList;
  public static void main(String[] args) {
    Student student1;
    RosterV3 roster = new RosterV3();
    student1 = roster.findStudent("2334");
    if(student1 == null)
        System.out.println("Student with id 2334 not found in class");
     else
        student1.printNeatly();
  }
  public RosterV3(){
    int classSize, i;
    classSize = Integer.parseInt(
         JOptionPane.showInputDialog(null,
             "Enter number of students in class"));
    studentList = new Student[classSize];
    for(i=0; i<classSize;i++)
        studentList[i] = new Student();
  }

  public Student findStudent(String id){
    . . .
    for(i=0; i < studentList.length;i++)
      if(id.equals(studentList[i].getId()))
         return studentList[i];
    return null;
  }
}
```

studentList is an instance data member.

Create a RosterV3 object.

Call a class method on object.

Constructor

Directly access instance data member.

16

Static Initializer

- Earlier, we initialized static variables upon declaration. This initialization takes place when the class is loaded.
  - Imported or used for the first time in a program.
- What if we want to do more?
  - E.g. set the initial value based upon user input?
- We can define a static initializer segment that gets executed only once when a class is loaded.

Static Initializer

```java
class Student {
  static int nextID;
  private String name, id;
  private double gpa;

  static {
    String str = JOptionPane.getInputDialog(null, "Enter starting ID");
    nextID = Integer.parseInt(str);
  }

  public Student(){
    id = "PUID" + Student.nextID;
    Student.nextID++;
    name = "Unknown";
    gpa = 0.0;
  }
  . . .
}
```

Static Initializer

- As with static methods, we cannot reference any non-static method or data member from the static initializer block.

PURDUE
UNIVERSITY

Reserved Word **this**

- Each object has a special data member called this that references the object itself.
- This keyword is useful for three purposes:
  - Accessing data members of the object
  - Accessing instance methods
  - Calling one constructor from another

PURDUE
UNIVERSITY

Monday, October 15, 12

Using this to Refer to Data Members

```java
class Student {

    String name;

    public void setName(String newName) {
        this.name = newName;
    }
    . . .
}
```

Usually implied and omitted.

```java
class Student {

    String name;

    public void setName(String newName) {
        name = newName;
    }
    . . .
}
```

© Sunil Prabhakar, Purdue University

Referencing Data Members

```java
class Student {

    String name;

    public void setName(String name) {
        name  = name;
    }
    . . .
}
```

Doesn't work as expected since the formal parameter masks the data member.

```java
class Student {

    String name;

    public void setName(String name) {
        this.name  = name;
    }
    . . .
}
```

Achieves the desired effect.

Monday, October 15, 12

Overloading

- Java allows multiple methods to share the same name
  - as long as they have different method signatures (i.e., different types of arguments, in order)
  - this is called overloading
- Which version is executed when called?
  - the one that matches the signature of the call.
- Method signature:
  - method name, and order of types taken as input
  - does not include return type

Monday, October 15, 12

Method Signatures

```
public class Test {
  void methodA() {                          methodA()
    . . .
  }
  void methodA(int a){                      methodA(int)
    . . .
  }
  void methodA(int a, int b){               methodA(int, int)
    . . .
  }
  void methodA(int a, float b){             methodA(int, float)
    . . .
  }
  void methodA(float b, String s){          methodA(float, String)
    . . .
  }
}
```

PURDUE
UNIVERSITY

Monday, October 15, 12

Overloaded Methods

```
public class Test {
  void methodA() {

    . . .
  }

☺ void methodA(int a){

    . . .
  }

☺ void methodA(int a, int b){

    . . .
  }

☺ void methodA(int a, Student b){

    . . .
  }

☺ void methodA(Student b, int a){

    . . .
  }

}
```

```
public class Test {
  void methodA() {

    . . .
  }

☺ void methodA(int a, int b){

    . . .
  }

☹ void methodA(int b, int a){

    . . .
  }

☹ int methodA(int c, int d){

    . . .
  }
☹ int methodA(){

    . . .
  }
}
```

24

Monday, October 15, 12

Overloaded Constructor

- The same rules apply for overloaded constructors
  - multiple constructors can be defined as long as each has a unique signature.

```java
public Student( ) {
    ...
}
public Student(String name) {

    ...
}
public Student(String name, double gpa) {

    ...
}
```

Monday, October 15, 12

Calling one Constructor from Another

- One can call a constructor of the same class from another using the keyword this.

```
static int  nextID;

public Student( ) {
    this("Unknown Name");
}
public Student(String name) {

    this(name, 0.0);

}
public Student(String name, double gpa) {

    this.id = "PUID" + Student.nextID++;

    this.name = name;

    this.gpa = gpa;

}
```

Monday, October 15, 12

Overloading and Type Casting

- When calling a method, if no matching signature is found, the compiler will attempt a match with automatic type casting.

```java
byte b;
methodA(b);
```

- Can match any of these methods:

```java
public methodA (short val) { ... }
public methodA (int val) { ... }
public methodA (long val) { ... }
public methodA (float val) { ... }
public methodA (double val) { ... }
```

```java
public methodA (Short val) { ... }
public methodA (Integer val) { ... }
public methodA (Long val) { ... }
public methodA (Float val) { ... }
public methodA (Double val) { ... }
```

Monday, October 15, 12

Copy constructor

- A copy constructor can be very handy.
- It takes an object as input and creates another object (of the same class) and copies the values.
- Useful also for preventing surreptitious access to private objects.
- If a method returns a pointer to a private object, then the client can modify the private object!
- Avoid this by returning a copy object

Use of a copy constructor

```java
class Jedi {
    private Person father;

    public void Jedi(Person f){
        father = f;
    }

    Person getFather(){
        return father;
    }
}
```

```java
class Corruptor {
    Jedi luke;
    Person p;
    public static void main(String[] args){
        luke = new Jedi(new Person("Unknown"));
        p = luke.getFather();
        p.setName("Darth Vader");
        p = luke.getFather();
        System.out.println(p.getName());
    }
}
```

```java
class Jedi {
    private Person father;

    public void Jedi(Person f){
        father = f;
    }

    Person getFather(){
        Person x;
        x = new Person(father);
        return x;
    }
}
```

© Sunil Prabhakar, Purdue University

PURDUE
UNIVERSITY

Monday, October 15, 12

Multiple classes per file

- So far, we only defined a single class per file.

- It is possible to define multiple classes in a single file.

- However, only one class can be public. This class must have the same name as the file (.java).

- Only public classes can be imported.

© Sunil Prabhakar, Purdue University

Monday, October 15, 12

Files and Classes

```java
class CS180Staff {
    . . .
}
class CS180Student {
    . . .
}
```

File: **Test.java**

javac Test.java

Two class files are created. Both are available to other classes in the same directory.

```
0101110010101....
```

File: **CS180Staff.class**

```
11100101111010....
```

File: **CS180Student.class**

PURDUE
UNIVERSITY

Monday, October 15, 12

Organizing Classes into a Package

- For a class, A, to use another class, B,
  - their byte code files must be located in the same directory, OR
  - Class B has to be organized as a package and A must import class B
- A *package* is a collection of java classes that can be imported.
  - E.g., javax.swing.* contains numerous classes that we import.

Monday, October 15, 12

Creating a package

- To create the CS180 package with CS180Staff as an included class:
  - Add **package** `CS180`; at the top of CS180Staff.java
  - CS180Staff must be a **public** class
  - Compile CS180Staff and place the .class file into the CS180 directory
  - Add the path to CS180 to the CLASSPATH variable

Monday, October 15, 12

Creating a package

```
package CS180;
public class CS180Staff {
    public static void main (String[] arg){
        ...
    }
}
```
File: **CS180Staff.java**

```
import CS180.*;
class MyClass {
    public static void main (String[] arg) {
        CS180Staff t;

            …
    }
}
```
File: **MyClass.java**

javac CS180Staff.java

CS180 directory

```
0101110010101....
      main
```
File: **CS180Staff.class**

CS180 package directory must be in CLASSPATH variable when running MyClass.

Files and Classes

```
class CS180Staff {
    public static void main (String[] arg){
        ...
    }
}
class CS180Student {
    public static void main (String[] arg){
        ...
    }
}
```

File: **Test.java**

javac Test.java

Two class files are created. Both can be executed with java ...

java CS180Staff

```
0101110010101....
        main
```

File: **CS180Staff.class**

java CS180Student

```
11100101111010....
        main
```

File: **CS180Student.class**

© Sunil Prabhakar, Purdue University

Monday, October 15, 12

Multiple Classes in a File

```java
package CS180;
public class CS180Staff {
    public static void main (String[] arg){
        ...
    }
}
class CS180Student {
    ...
}
```
File: **CS180Staff.java**

```java
import CS180.*;
class MyClass {
    public static void main (String[] arg) {
        CS180Staff t;
        …
    }
}
```
File: **MyClass.java**

javac CS180Staff.java

CS180 directory

```
0101110010101....
    main
```
File: **CS180Staff.class**

```
11100101111010....
    main
```
File: **CS180Student.class**

Only CS180Staff can be imported and used outside the package directory.

CS180Student can only be used within the CS180 directory.

CS180 package directory must be in CLASSPATH variable when running MyClass.

© Sunil Prabhakar, Purdue University

Monday, October 15, 12

The null constant

- **null** is a special value. Its type is that of a reference to an object (of any class).
- We can set an object identifier to this value to show that it does not point to any object.
  - Student student1=**null**;
- A method that returns objects (of any class) can return a **null** value.
- Note that you will get a run-time error if you access a data member or call a method on a **null** reference -- *null pointer exception*.