COMP 250

Lecture 8

Objects & Classes 3:

packages,

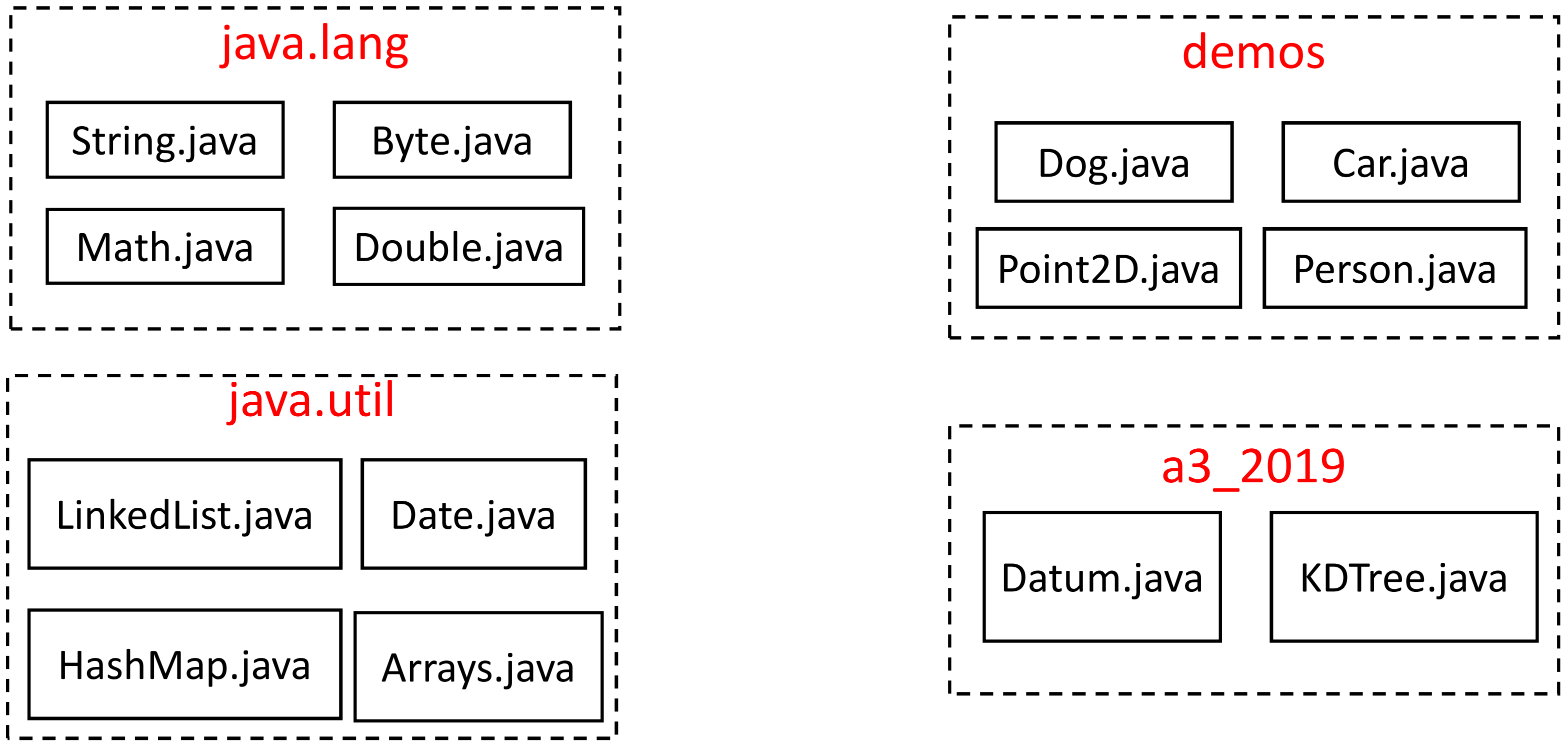
access modifiers: `public, private`

UML class diagram

Jan. 24, 2022

Packages (recall lecture 4)

A package is a set of classes. The two on left are examples from the standard Java library. The two on right are examples of my own packages.



We put a package statement at the first line of our class definition file.
This says which package the class belongs to.

Point2D.java

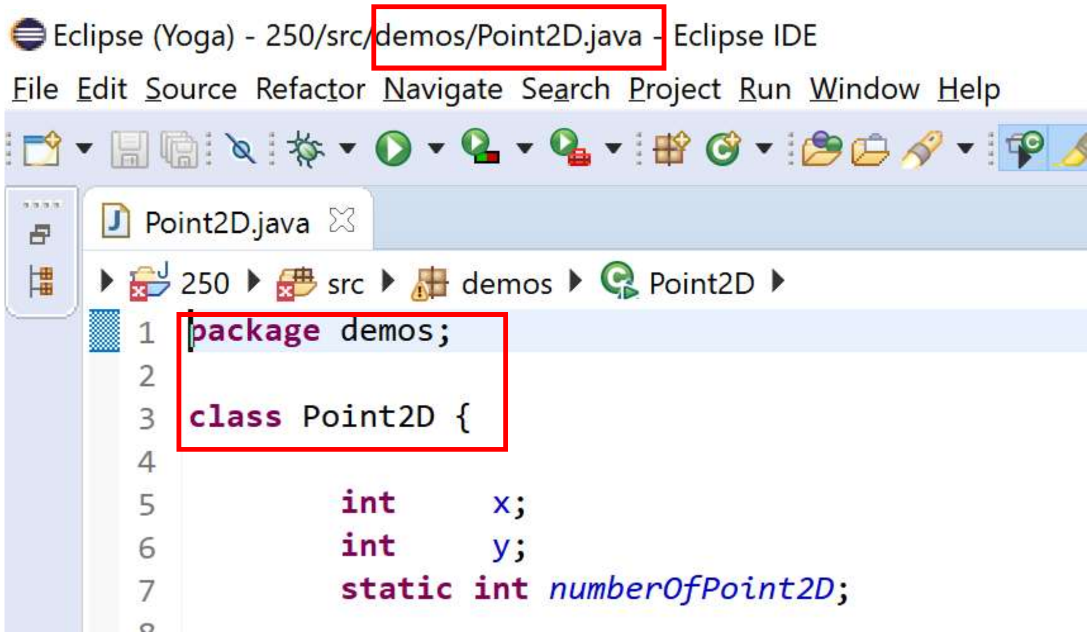
```
package demos;
class Point2D{
             :
}
```

Packages and File Folders (recall lecture 4)

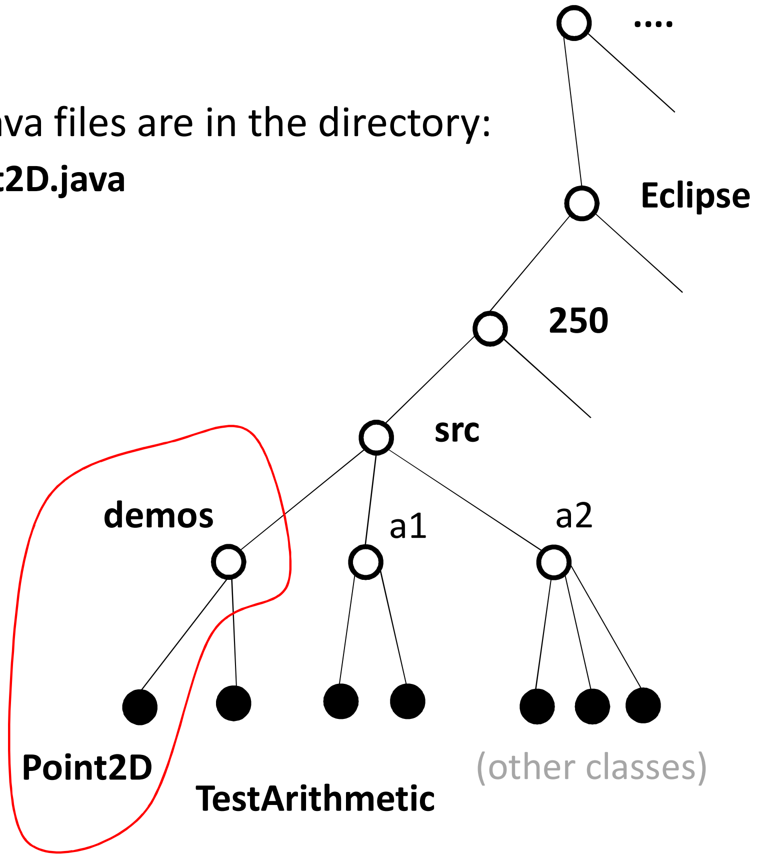
Packages are organized as folders on your computer file system.

In this Eclipse example, there is a project name (“250”) and the .java files are in the directory:

C:\Users\MichaelLanger\Dropbox\Eclipse\250\src\demos\Point2D.java


Eclipse (Yoga) - 250/src/demos/Point2D.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Point2D.java
250 ▸ src ▸ demos ▸ Point2D ▸

```
package demos;

class Point2D {

        int     x;
        int     y;
        static int numberOfPoint2D;
```



Accessing a class in another package

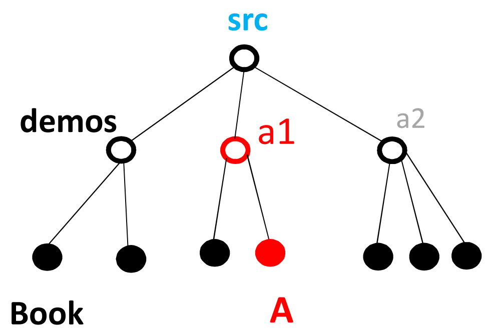
If class A wants to construct or reference a class B object and if class B is in a different package, then class A must tell the compiler where to find class B.

It can do so in three ways:

1) Specify the entire path.

For example, suppose class A is in the `a1` package and it wants to reference `Book` from the `demos` package. Then, it can *fully specify* the class name (relative to src).

```
demos.Book    myBook = new demos.Book();
```



2) Include an import statement (in class A), namely `import class B`*:*

e.g. **`import demos.Book;`**

The import statement comes *after* the package statement. It tells the compiler that the class `Book` is found in the package `demos`.

```
Book     myBook = new Book();
```

Advantage of import: it saves typing the full class name.

Disadvantage of import: when we read code locally and we see a class name (e.g. variable type declaration), we won't necessarily know what package that class belongs to.

3) Import an entire package. Example (in class A):

```
import  demos.*;
import  java.util.*
```

Class A now can refer to any class (B) that is in one of these imported packages.

No, there cannot be name conflict. e.g. you cannot import a class B from `demos` when compiling a class whose package has a class with the name B.

Automatic imports

For convenience, the Java compiler automatically imports all classes from two packages:

- the *current* package

- the `java.lang` package

The latter contains classes `Math`, `String`, … so no import statement is need to use these classes.

The discussion on the previous slides suggests that if you create a class A, you can access any other class B just by specifying its full path or by importing it, namely you can construct and reference objects of class B, and invoke their methods.

That's not the whole story, however.

Each class (B) also needs to define where it and *each of* its fields and methods is visible.

There are three levels of visibility: (and a fourth level that I will mention in a few weeks.)

- visible only from within that class (B) (`private`)
- visible from any class A within the same package (by default)
- visible from any class A in *any* package (`public`)

Visibility (or Access) modifiers

Suppose an instruction in class A refers to a field or method in class B (static or not)…

Visibility modifier in B	**A = B**	**A & B in same package**	**A & B in different packages**
`public`	✓	✓	✓
(package)	✓	✓	✗
`private`	✓	✗	✗

Classes can have either `public` or the default (package) modifier, but not `private`.

There is also a `protected` modifier, which I will mention in a few weeks.

To specify “package visibility”, we don’t use any modifier at all, as in most of the examples in previous lectures.

The keyword `package` is used instead for stating the package name of the class.

Dog.java

```
package demos;

class Dog {
   ⋮
}
```

If we wrote `public` here, then the class `Dog` would be visible from *any* package.

Examples….

Dog.java

```
package demos;

class Dog {
    ⋮
}
```

Owner.java

```
package demos;

class Owner {
    Dog d;
    ⋮
}
```

Does the compiler allow this declaration ?

- ➢ Yes, because class `Dog` has package visibility and both classes are in the same package.

Examples….

package visibility

Dog.java

```
package demos;

class Dog {
    ⋮
}
```

Tester.java

```
package a1;
import demos.Dog;

class Tester {
    Dog d;
    ⋮
}
```

Does the compiler allow this ?

➢ No, because class `Dog` has package visibility only.

(The error is in the import statement.)

Examples….

Dog.java

```
package demos;

public class Dog {
    ⋮
}
```

Tester.java

```
package a1;
import demos.Dog;

class Tester {
    Dog d;
    ⋮
}
```

Does the compiler allow this ?

- Yes, because class Dog is declared to have public visibility and Dog is imported.

Examples....

Dog.java

```
package demos;

public class Dog{
     public String name;

          :
}
```

Tester.java

```
package a1;
import demos.Dog;

class Tester {
     public static void main(…){

       Dog  myDog = new Dog();
       myDog.name = "Buddy";
}
```

Does the compiler allow this ?

➢ Yes, since `name` has `public` visibility.

Examples….

Dog.java

```
package demos;

public class Dog{
     private String name;

          :
}
```

Tester.java

```
package a1;
import demos.Dog;

class Owner {
     public static void main(…){

       Dog  myDog = new Dog();
       myDog.name = “Buddy”;
}
```

Does the compiler allow this ?

- No, since field `name` has **`private`** visibility (even though the `Dog` class is public).

Examples….

Dog.java

```
package demos;

public class Dog{
  private String name;

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

Tester.java

```
package a1;
import demos.Dog;

class  Tester {
     public static void main(…){

        Dog  myDog = new Dog();
       myDog.setName(“Buddy”);
}
```

Does the compiler allow this ?

- Yes, since method `setName` has **`public`** visibility (even though the field `name` is private).

Examples….

Dog.java

```
package demos;

public class Dog{
  private String name;

  private void myHelper(){
      //  something useful
  }
}
```

Tester.java

```
package a1;
import demos.Dog;

class  Tester {
     public static void main(…){

       Dog  myDog = new Dog();
       myDog.myHelper( );
}
```

Does the compiler allow this ?

- No, since method `myHelper` has **`private`** visibility.

Exercise

The `Point2D` class was previously defined with package visibility.

Create a second class – say `Test` -- in a *different* package. Give it the `main` method below.

Correct the compile time errors. You will need to change both classes. Do it two ways, namely with or without using an import statement.

```
public static void main(String[] args) {

      Point2D  p1 = new  Point2D( 23, 85 );
      Point2D  p2 = new  Point2D( 5,  6 ) ;

      System.out.println(  distanceBetween(p1, p2) );
      System.out.println(  p1.distanceTo( p2 ) );
}
```

Getter and Setter Methods (*for "Encapsulation")*

"getters" = "accessors"   // don't change field values

"setters" = "mutators"   // do change field values

Class fields (static or non-static) are *typically* private, although they *are allowed* to be public or package visible.

In the Java API,   only public fields and methods are listed. The fields are often constants e.g. Math.PI

The Math class is `final`. I will cover the `final` modifier later.

Motivation for getters and setters

Suppose some Java application allows users to enter their first and last name.

```
public  class   User {
    public  String  lastName;
    public  String  firstName;

    User(String first, String last){
          firstName = first;
          lastName = last;
    }
}
```

Q: What is the problem with the above ?

A: We don’t want to allow this (in another class).

```
User   u  =  new User(“Sue”,”Lin”);
u.lastName  =  “&$(!”;
u.firstName =  “---!”;
```

and we don’t want this

```
User   u  =  new User(“!!!”,”?*X”);
```

A better approach is to control what users are allowed to enter as their first and last name.

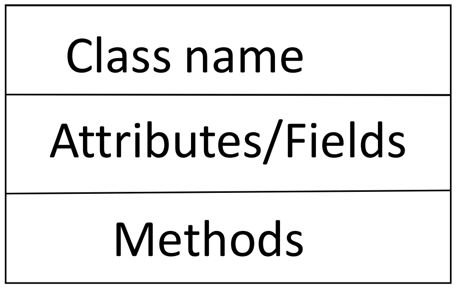
```
public  class   User {
           private  String  lastName;
           private  String  firstName;

           User(String  last,  String first){
              //   call setters
                   setLastName( last );          //  these methods also can be called
                   setFirstName( first );        //  without the constructor
           }

           public setFirstName( String first){
             //  This method verifies that the first name satisfies certain rules.

           }
           public setLastName( String last){
             //  This method verifies that the last name satisfies certain rules.
           }
}
```

UML Diagrams (intro only)

Unified Modeling Language (UML) provides a set of standard diagrams for graphically depicting what is in a class, and relationships between classes.

This is a central topic in COMP 303 Software Design.

Here we'll briefly discuss just how to represent what's in a class.



Example – Dog Class

Fields/Attributes

- String name
- Person owner
- static int numDogs

Constructors

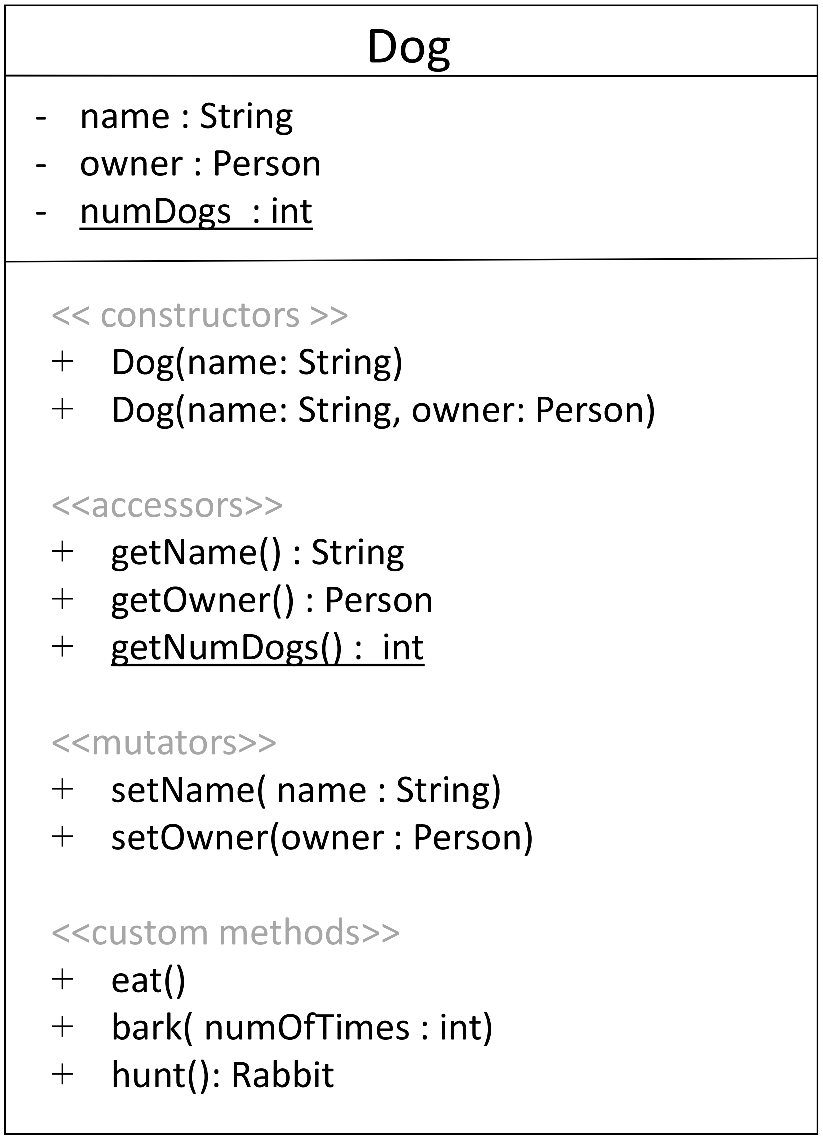
- Dog(String name)
- Dog(String name, Person owner)

Accessors and Mutators

- getName
- getOwner
- setName
- setOwner
- static getNumDogs

Other Methods

- eat()
- bark()
- hunt()



```
Dog
-   name : String
-   owner : Person
-   numDogs  : int

<< constructors >>
+   Dog(name: String)
+   Dog(name: String, owner: Person)

<<accessors>>
+   getName() : String
+   getOwner() : Person
+   getNumDogs() :  int

<<mutators>>
+   setName( name : String)
+   setOwner(owner : Person)

<<custom methods>>
+   eat()
+   bark( numOfTimes : int)
+   hunt(): Rabbit
```

+ public
- private

The type is listed *after* the variable name.

Static fields and methods are underlined.

ASIDE: I will use UML diagrams sometimes, but I will not examine you on them.

Coming up…

Lectures

Wed. Jan. 26

ArrayLists

Fri. Jan. 28

Singly Linked Lists

Assessments

Fri. Jan. 28

Quiz 1 (lectures 1-7, including the leftover part at start of today's video)

- practice quiz posted today

Assignment 1 to be posted

- you will have 2 weeks to do it