COMP 250

Lecture 36

modifiers
- public, private
- static, final

Dec. 1, 2017
Packages

java.lang
- Object.java
- String.java
- Math.java

java.util
- LinkedList.java
- HashMap.java

lectures
- Dog.java
- Beagle.java

a4
- MyHashTable.java
visibility/access modifiers

• public
• package    (default, so not a reserved word)
• protected   (not discussed today)
• private

These modifiers can be for a class, or a class member i.e. method or field.
Q: Does the compiler allow this?
A:
Q: Does the compiler allow this?
A: Yes, because Dog is public.
I did not write visibility modifier(s) on right side because they are irrelevant here. I am asking about the visibility of the Dog class on the left.
Q: Does the compiler allow this?
A:
Q: Does the compiler allow this?
A: No, because Dog class has package visibility only.
Q: Does the compiler allow this?
A:
Q: Does the compiler allow this?
A: Yes, because Dog class has public visibility.
Q: Does the compiler allow this?

A:
Q: Does the compiler allow this?
A: No, because `Dog` class has package visibility only.
public class Dog{
    public String name;
    public Dog() {...}
}

public class Person {
    main() {
        Dog myDog = new Beagle();
        myDog.name = "Buddy";
    }
}

Q: Does the compiler allow this?
A:
public class Dog{
    public String name;
    public Dog() {...}
}

class Person {
    main() {
        Dog myDog = new Beagle();
        myDog.name = “Buddy”;
    }
}

Q: Does the compiler allow this?
A: Yes, since name is public.
Q: Does the compiler allow this?
A:
Q: Does the compiler allow this?
A: No, since Dog.name has package visibility only.
Q: Does the compiler allow this?
A: Yes, since Dog.name has package visibility and now the two classes are in the same package.
package lectures

public class Dog {
    private String name;
    public Dog() { ... }
}

Q: Does the compiler allow this?
A: Yes.
package lectures
public class Dog{
    private String name;
    public Dog() {...}
}

package lectures
class Person {
    :  
    main() {
        Dog myDog = new Beagle();
        myDog.name = "Buddy";
    }
}

Q: Does the compiler allow this ?
A: No, since name is private.
Getter and Setter methods

Java class fields are typically private.

Getters = “accessors” // don’t change field values

Setters = “mutators” // change field values
public class Dog {
    private String name;

    public Dog() {
    }

    public String getName(String name) {
        this.name = name;
    }

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

This is the typical way to do things.
Q: What is the problem with making the field public?

How could it possibly matter which of these you use?

```java
public class Dog {
    public String name;

    public Dog(){ }
}
```

```java
public name;

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

    public Dog() {
    }
}

Q: What is the problem with making the field public?

A: If you use the setter on the previous slide, then it doesn’t matter!

But suppose some client program has the following:

Dog myDog = new Dog();
myDog.name = "&$(!";

You probably don’t want to allow that.
public class Dog {
    private String name;

    public Dog() {
    }

    public void setName(String name) {
        // verify that name obeys some rules
    }

    public String getName(String name) {
        // now needed also
    }
}

Thus, using setter methods gives you more control over what clients can do.

And if you make the field private, then you need a getter too.
public class Beagle {
  private Person owner;

  Beagle() { ....}

  public Rabbit rabbit = new Rabbit();

  public Person getOwner(){
    return owner;
  }
}
Ever noticed...? You cannot define visibility modifiers for a local variable within a method.

```java
public class Beagle {
  private Person owner;

  Beagle() { 
  }

  public hunt(){
    Rabbit rabbit = new Rabbit();
    :
  }

  public Person getOwner(){
    return owner;
  }
}
```

It is a scope issue. Other methods in this class (or other classes) cannot reference this variable.
Beagle object
Person owner

Class Descriptors

Methods are here

Dog
class descriptor

Beagle
class descriptor

TestProgram
class descriptor

Call Stack

Local variables and parameters of methods are here

hunt()
this
Rabbit rabbit

Test.main()
visibility and inheritance
Q: Which of the variables in class A are inherited by class B?

A: All three x, y, z are inherited.
public class A {
    public int x;
    int y;
    private int z;
}

class B extends A {
    B( ..., int z) {
        this.z = z;
    }
}

Q: Does the compiler allow this?

A: No. Variable z is not visible in B since it declared as private in A.
public class A {
    public int x;
    int y;
    private int z;

    A(){ ... };

    public int getZ{
        return z;
    }

    public void setZ( int z){
        this.z = z;
    }
}

class B extends A {
    B( ..., int z) {
        setZ(z);
    }
}
We can extend any public class (even across packages)
We can extend any **public** class (even across packages)

Not allowed, since WordTree isn’t visible.
class Outer

public int x
package int y
private int z

private class Inner

private double x

Outer and Inner have access to all fields of each other.

*Details omitted.*
COMP 250

Lecture 36

modifiers
- public, private
- static, final

Dec. 1, 2017
static fields and methods

class Dog{
    String name;
    static int numDogs = 0;  // a.k.a. class field

    Dog(){ ..
        numDogs ++;
    }

    static int getNumDogs() {
        return numDogs;
    }

}
Class Descriptors

All methods are here

Static fields are here.

Dog

class descriptor

static int numDogs
static Dog queenDog
static Dog kingDog

Beagle

class descriptor

TestProgram

class descriptor

Call Stack

Local variables and parameters of methods are here

bark()

Test.main()

Objects

Instance fields are here

Beagle object

other objects
A method is **static** if it is not invoked by an object.

class Test {
    public int methodA() { ..... }
    public static void main() { ... }
}

final modifier

```java
final class Dog{
    :
}

class Beagle extends Dog { // not allowed
    :
}

e.g. Java.lang.{String, Math, ..}
```
class Dog{
    void final bark(){ ...
}
}

class Beagle extends Dog {
    void bark(){ ....  }
    // not allowed
    // compiler error
}
class Dog{
    Dog final myDog;

    public static void main(){
        myDog = new Beagle(“Buddy”);
        :
        myDog = new Poodle(“Willie”);
        // not allowed (compiler error)
    }
}
static + final modifiers

```java
public final class Math{  // static classes only allowed
  // as inner classes

  public static final double PI = 3.14...;
  public static final double E = 2.71...;

  public static double sqrt( double x) { ....}
}
```

// We need to say Math.PI since there is no Math object.
Next Week

• Monday : Grad School

• Wed./Thurs. : Final exam & Beyond COMP 250