Modifiers
- public, private, ...
- final, static, ...

Packages
- `java.lang`:
  - `Object`, `String`, `Math`, ...
- `java.util`:
  - `LinkedList`, `HashMap`:
- Lectures:
  - Dog, Beagle
- `a3`:
  - TriNode, Trie

Visibility/access modifiers
- public
- [default]
- protected
- private

You write a class and you want to restrict access to class or to particular member fields & methods.

If we drop the modifier public then visibility is restricted to classes within the same package
⇒ Compiler gives an error.

Same problem:
If we drop the modifier public then visibility is restricted to classes within the same package
⇒ Compiler gives an error.
Suppose class A is visible to class B.

e.g. "public class A"
    "class A" and A and B
    are in same package

Q: can class B...
  - reference a field of A?
  - invoke a method of A?

\[ \Rightarrow \text{Dog.name is visible to Person.} \]

Suppose Dog is visible to Person

(review from Comp 202)

A

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

B

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

Compiler error.
(Dog.name not visible to Person.)

package lectures

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

package lectures

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

Same package
\[ \Rightarrow \text{Dog.setName()} \text{ is visible to Person.} \]

Dog class is visible to Person class.

package lectures

public class Dog{
private String name;
public Dog(){...} ...
private void setName(String name){...} ...
String getName(){... return name;...}
}

package lectures

public class Person{
main() {
  Dog myDog;
  myDog = new Beagle();
  myDog.setName("Bud")
  myDog.setGetname();
}

package a

public class A
private x

package b

extends

can access x

can access y

cannot access z

cannot access y

package c

extends

can access x

cannot access y

cannot access z

cannot access y

Dog: setName(...) is not visible to Person.
Dog: getName(...) is visible to Person.
public static class MyClass {
  
}

This produces a compiler error.

```java
class Dog {
  String name
  static int numDogs = 0
  Dog () {
    numDogs ++
  }
  static int getNumDogs () {
    return numDogs
  }
}
```

Static members: fields and methods

```
public static void main(---) {
  
}
```

Why? You need to be able to run this method without an instance.
**final class modifier**

```java
package lectures;

final class Dog {
    ...
}
```

Compiler error. (Cannot extend a final class)

e.g. `Java.lang.String, Math`

**final method modifier**

```java
package lectures;

class Beagle extends Dog {

    void final bark() {
        ...
    }
}
```

Compiler error. (Cannot override a final method)

Note: private methods are automatically final.

**Static + final field modifier**

```java
public final class Math {
    public static final double PI = 3.14
    public static final double E = 2.71
    ...
    public static double sqrt(double x) {
        ...
    }
}
```

**Class MyClass**

```java
void makeDogs (int n) {
    ArrayList<Dog> dogs;
    for (int k=0; k<=n; k++) {
        dogs.add(new Dog());
    }
}
```

Note: method parameters and local variables do not have modifiers.