Announcements

Project 2. Some tips:

- Lists, Sets, Maps. Use them (not necessarily all of them).
- If you find yourself having nested generic declarations, e.g. Map<Map<... or List<List<..., this is fine, but consider creating a new class that has an instance variable that hides the ugly truth.
  - E.g. instead of having x.get(5).get(3) for a list of lists, you can write your own class that has a get(5, 3) method.
- Break things down into smaller pieces. Layers of abstraction are absolutely vital.
  - Trying to do everything at the lowest layer (TETile[][]) will probably be very unpleasant.
  - Example: A bad idea I briefly had was a single method with tricky nested for loops that tried to draw L-shaped hallways.
- Recursion: Very natural for world generation (but not necessary).

datastructur.es

Announcements

Project 2.

- If you had any sort of partner matching mishap (e.g. partner never replied) and you want a partner, please fill out the partner matching form again ASAP.

datastructur.es

CS61B: 2018

Lecture 15: Packages, Javadocs, Access Control, Objects

- Packages and JAR Files
- Javadocs
- Access Control
- Object Methods: Equals and toString()

datastructur.es

Packages and JAR Files

datastructur.es

Packages

To address the fact that classes might share names:

We won't follow this rule. Our code isn't intended for distribution.

- A package is a ***namespace*** that organizes classes and interfaces.
- Naming convention: Package name starts with website address (backwards).

```
package ug.joshh.animal;

public class Dog {
    private String name;
    private String breed;
    private double size;
```

Dog.java

If used from the outside, use entire ***canonical name***.

```
ug.joshh.animal.Dog d =
    new ug.joshh.animal.Dog(...);
```

```
org.junit.Assert.assertEquals(5, 5);
```

If used from another class in same package (e.g. ug.joshh.animal.DogLauncher), can just use ***simple name***.

datastructur.es

Creating a Package

Two steps:

- At the top of every file in the package, put the package name.
- Make sure that the file is stored in a folder with the appropriate folder name. For a package with name `ug.joshh.animal`, use folder ug/joshh/animal.

```
package ug.joshh.animal;

public class Dog {
    private String name;
    private String breed;
    private double size;
```

Dog.java

If used from the outside, use entire ***canonical name***.

```
ug.joshh.animal.Dog d =
    new ug.joshh.animal.Dog(...);
```

```
org.junit.Assert.assertEquals(5, 5);
```

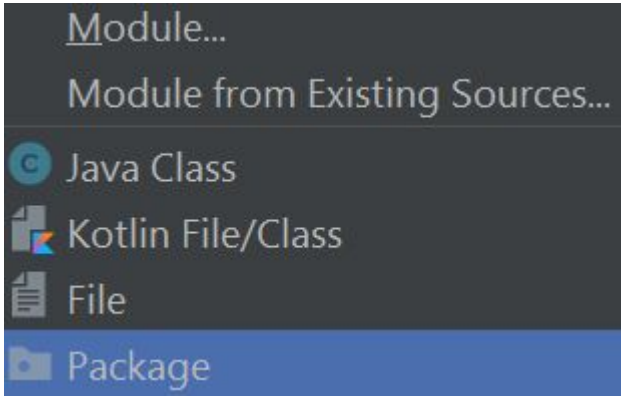
If used from another class in same package (e.g. `ug.joshh.animal.DogLauncher`), can just use ***simple name***.

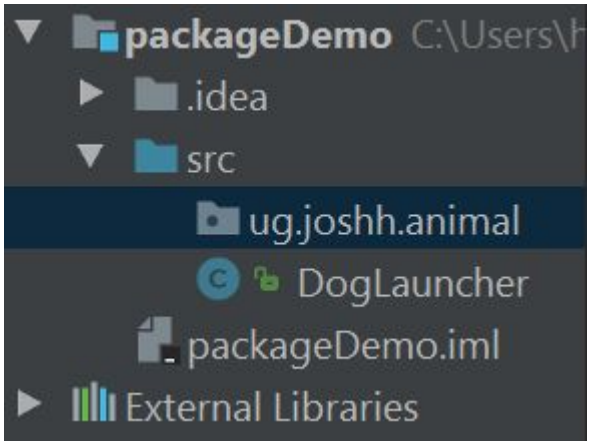
datastructur.es

Creating a Package in IntelliJ

In Intellij:

- File -> New Package
- Choose a package name, e.g. “ug.joshh.animal” and it will appear in the list of files for your project.
  - In the example to the right, the project has the empty package `ug.joshh.animal`, and a source file called `DogLauncher` that is not part of the `ug.joshh.animal` package.

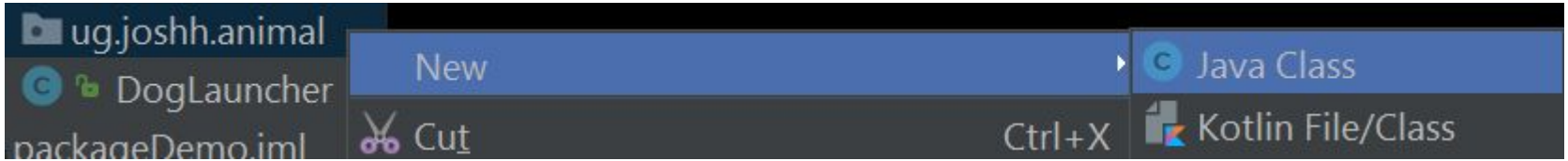




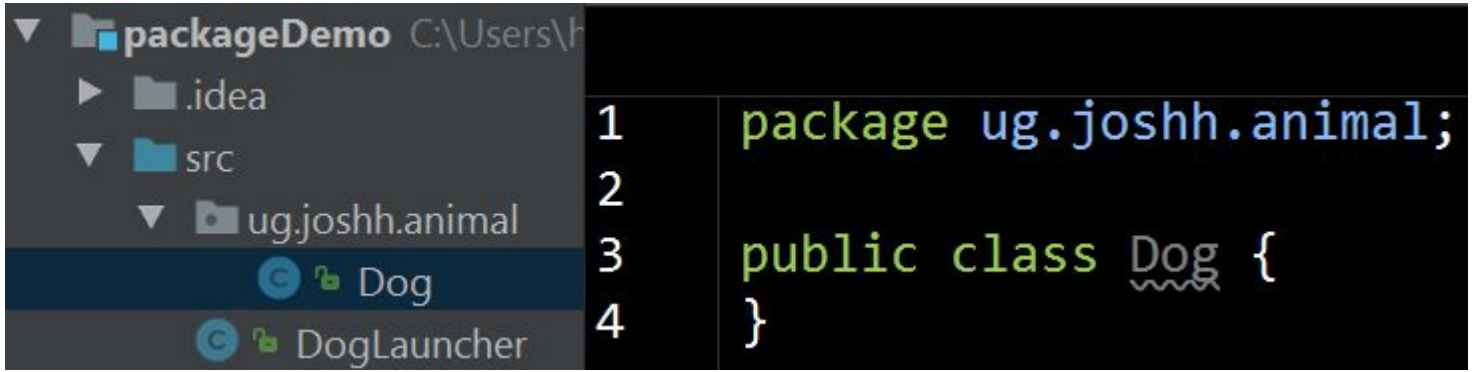
datastructur.es

Adding a Java File to a Package in Intellij

Right-click the package name, and select New → Java Class.



This will create a file called Dog.java that is under the subheading of "ug.joshh.animal", and will automatically fill in the first line of the file with the package declaration.



datastructur.es

What's Happening Under the Hood

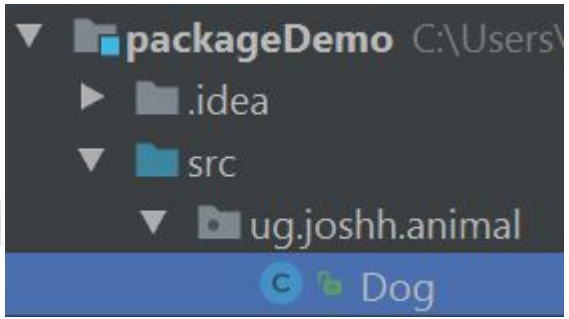
There are two steps to creating a package:

- At the top of every file in the package, put the package name.
- Make sure that the file is stored in a folder with the appropriate folder name. For a package with name `ug.joshh.animal`, use folder ug/joshh/animal.

Under the hood the things we did in IntelliJ did exactly this!

- To see this in action, right click the Dog class in the top left, and then click "Show in Explorer".
- You'll see that the Dog.java file is in a folder called

  …./src/ug/joshh/animal



datastructur.es

Using Packages

As we saw in a previous lecture, to use a class from package A in a class from package B, we use the ***canonical name***.

For example, in the `DogLauncher` class, which is not part of the ug.joshh.animal package, can create a Dog using the syntax below.

```
ug.joshh.animal.Dog d =
    new ug.joshh.animal.Dog(...);
```

By using an **import** statement, we can use the ***simple name*** instead.

```
import ug.joshh.animal.Dog;
...
Dog d = new Dog(...);
```

datastructur.es

The Default Package

Any Java class without a package name at the top are part of the “default” package. As Stack Overflow user Dan Dyer wisely puts it: “You should avoid using the default package except for very small example programs.”

- In other words, from now on, when writing real programs, your Java files should always start with a package declaration.
- Idea: Ensure that we never have two classes with the same name.

Note: You cannot import code from the default package!

- No way that anything in the `ug.joshh.animal` package could ever use `DogLauncher` (which is in the default package).

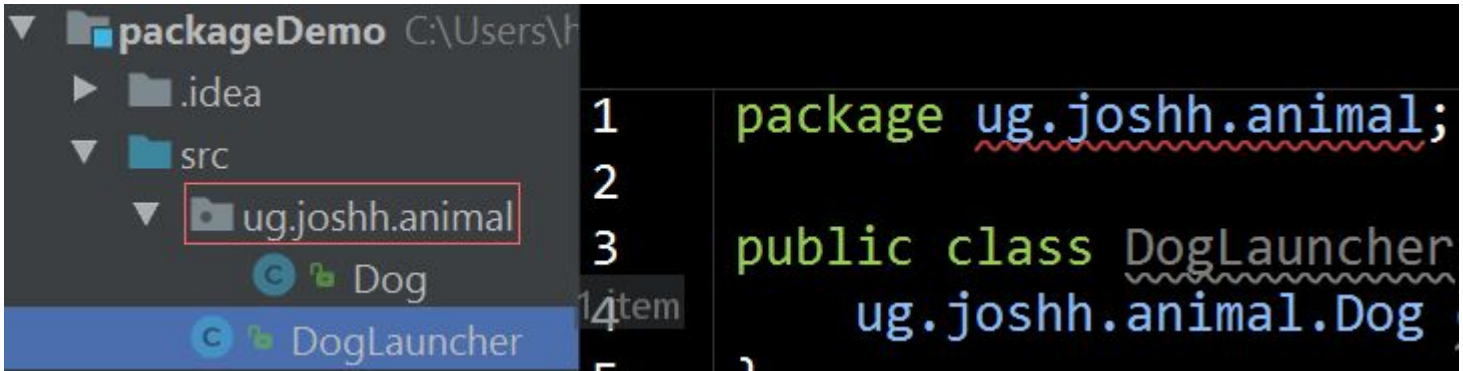
datastructur.es

Moving a Class into a Package

From this point on in 61B, all code for each HW and project will be part of its own package. You'll never need to define multiple packages at once in 61B.

If you realize you should have made something part of a package, simply:

- Add "package [packagename]" to the top of the file. IntelliJ will complain that the file is in the wrong folder and won't compile until you...
- Move the .java file into the appropriate folder.
  - Protip, you can drag the file in the top left Project viewer.



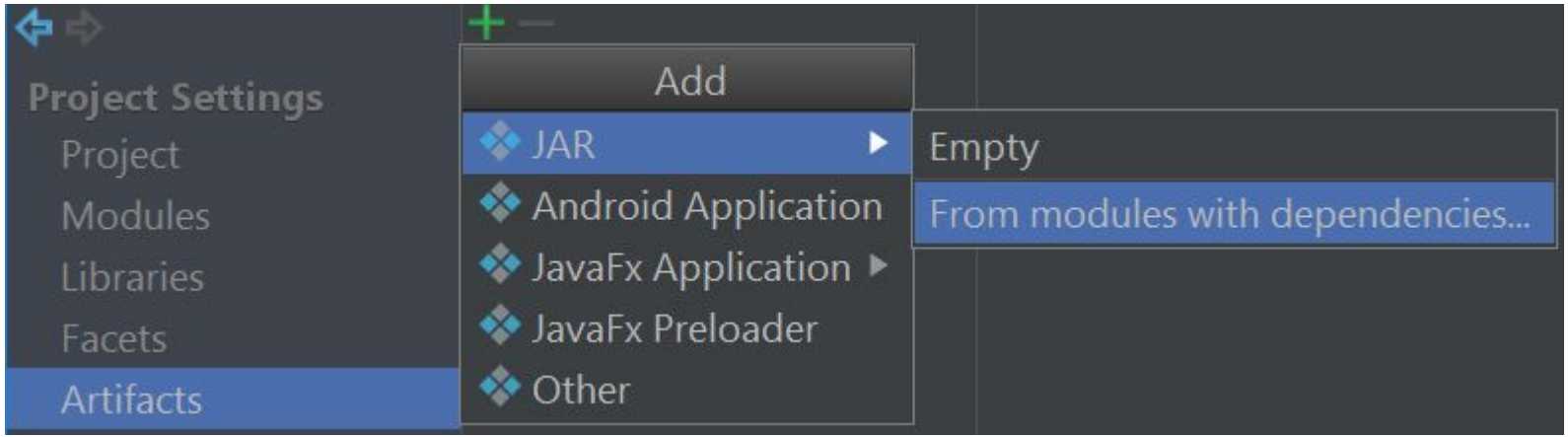
datastructur.es

JAR Files

Suppose you've written a program that you want to share.

- Sharing dozens of .class files in special directories is annoying.
- Can instead share a single .jar file that contains all of your .class files.
  - JAR files are really just zip files, but with some extra info added.

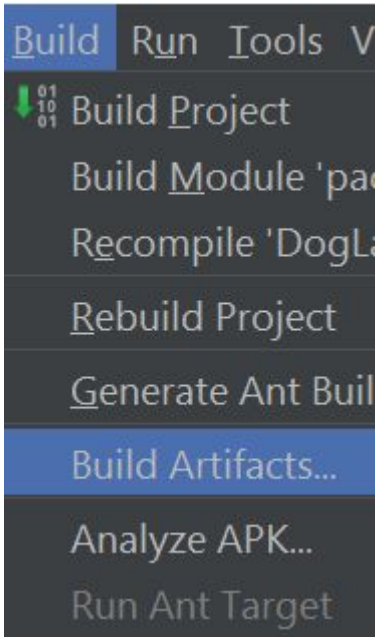
To create a JAR file: 1. Go to File → Project Structure, and select the option below:

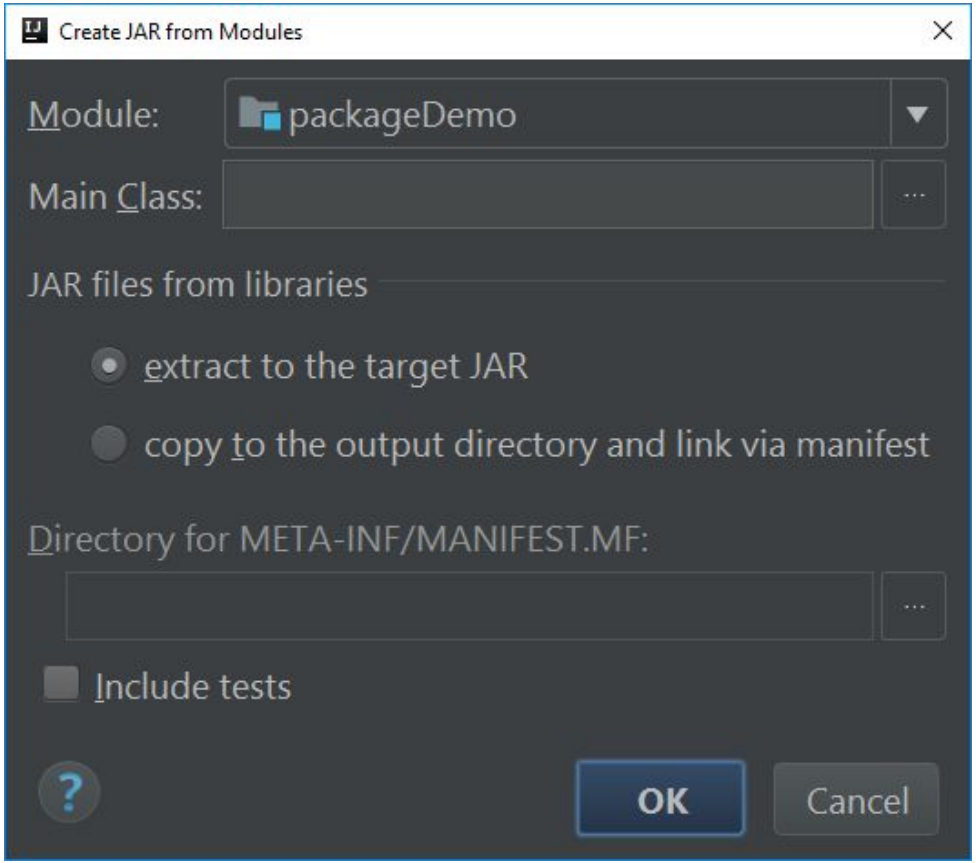


datastructur.es

JAR Files

Step 2: Click OK a couple of times.



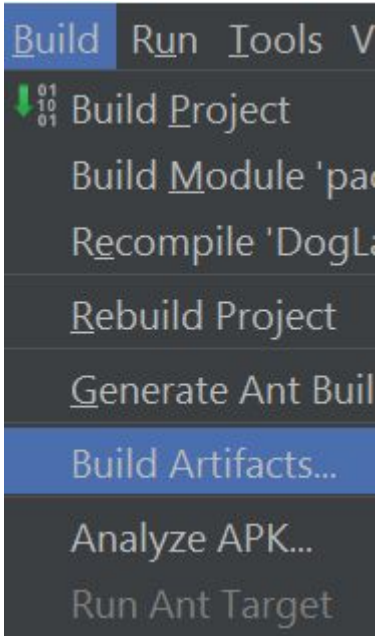


datastructur.es

JAR Files

Step 3: Click Build -> Build Artifacts.

- The jar file will be generated in a folder called “Artifacts”.



Step 4: Give to other Java programmers, who can use your jar by importing it into IntelliJ (or otherwise).

- Also possible to run a “Main class” of a package from command line. Not discussed in lecture. See Google for more.

datastructur.es

Note on JAR Files

JAR files are just zip files.

- They do not keep your code safe!
- Easy to unzip and transform back into .Java files.
- Important: Do not share .jar files of your projects with other students.

datastructur.es

Build Systems

To avoid the need to import a bunch of libraries, put files into the appropriate place, and so forth, there exist a number of “Build Systems”.

Popular build systems include:

- Ant
- Maven
- Gradle (ascendant)

Very useful for large teams and large projects. Advantages are fairly minimal in 61B. We’ll use Maven in Project 3.

datastructur.es

Access Control

datastructur.es

Access Control with Inheritance and Packages

How do public and private behave with packages and subclasses?

- Can a subclass access a private member of its superclass?
  - Could a `SpecialArrayMap` access the private `KeyIterator` class?
- Can a class X in a package access a private member of its package buddy Y?
  - Could a `GuitarString` access the private variables of an `ArrayRingBuffer`?

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
private	Y	N	N	N

No in both cases:

- Only code from the a given class can access private members.

datastructur.es

Access Control with Inheritance and Packages

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
private	Y	N	N	N

```
package syntax3.map;

public class ArrayMap ... {
   private int size; ...
```

```
package syntax3.map;

public class MyonicArrayMap extends ArrayMap {
   public MyonicArrayMap(int s) {
      size = s; ...
```

```
$ javac *.java
MyonicArrayMap:5: error: size has private access in ArrayMap
    size = s;
    ^
```

datastructur.es

The Protected Keyword

Protected modifier allows package-buddies and subclasses to use a class member (i.e. field/method/constructor).

- Outsiders cannot access.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
private	Y	N	N	N

```
package syntax3.map;

public class ArrayMap ... {
   protected int size; ...
```

```
package syntax3.map;

public class MyonicArrayMap extends ArrayMap {
   public MyonicArrayMap(int s) {
      size = s; ...
```

Access Control with Inheritance and Packages

Using no modifier allows classes from the same package, but not subclasses, to access the member.

- Often referred to as “package private”.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
	Y	Y	N	N
private	Y	N	N	N

datastructur.es

Access Control with Inheritance and Packages

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
	Y	Y	N	N
private	Y	N	N	N

```
package syntax3.map;
public class ArrayMap ...
   int size; ...
```

```
package syntax3.map;
public class MyonicArrayMap extends ArrayMap {
   public MyonicArrayMap(int s) {
      size = s; ...
```

Q: Will this work?

Yes!

```
package syntax3.map;
public class MapHelper {
   public static void printSize(ArrayMap am) {
      System.out.println(am.size);
```

Q: Will this work?

Yes!

datastructure.es

Random Question Out of Nowhere:

Which looks best? A/B/C? Answer here: http://shoutkey.com/out

A

```
String actual = MapHelper.maxKey(m);
String expected = "house";
assertEquals(expected, actual);
```

B

```
String actual = MapHelper.maxKey(m);
String expected = "house";
assertEquals(expected, actual);
```

C

```
String actual = MapHelper.maxKey(m);
String expected = "house";
assertEquals(expected, actual);
```

datastructur.es

Access Control with Inheritance and Packages: yellkey.com/any

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
	Y	Y	N	N
private	Y	N	N	N

```
package syntax3.map;
public class ArrayMap ... {
   int size; ...
```

```
package illuminati;
public class ChattyMap extends ArrayMap {
   public void printSize() {
      System.out.println(size); ...
```

Q: Will this work?

datastructur.es

Access Control with Inheritance and Packages

	Subclass	World
	Y	Y
	Y	N
	N	N
	N	N

```
$ javac ChattyMap.java
../map3/syntax3/*.java
ChattyMap:4: error: size is not public in
ArrayMap; cannot be accessed from outside
package
    System.out.println(size); ...
    ^
```

```
package syntax3.map;
public class ArrayMap ... {
   int size; ...
```

```
package illuminati;
public class ChattyMap extends ArrayMap {
   public void printSize() {
      System.out.println(size); ...
```

No!

Q: Will this work?

datastructur.es

A Point to Ponder

Why was Java designed to have the top table instead of the bottom one (or something similar)?

In other words, why are package members considered more “secret” than subclass members?

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
	Y	Y	N	N
private	Y	N	N	N

Modifier	Class	Package	Subclass	World
	Y	Y	Y	Y
	Y	Y	Y	N
	Y	N	Y	N
	Y	N	N	N

datastructur.es

A Point to Ponder

Why was Java designed to have the top table instead of the bottom one (or something similar)?

- Extending classes you didn't write is common.
- Packages are typically modified only by a specific team of humans.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
	Y	Y	N	N
private	Y	N	N	N

Modifier	Class	Package	Subclass	World
	Y	Y	Y	Y
	Y	Y	Y	N
	Y	N	Y	N
	Y	N	N	N

datastructur.es

The Default Package

What about code that doesn't have a package declaration?

- As mentioned earlier, code without a package label is part of an unnamed package, a.k.a. the "default package".

Don't have any access modifiers or package names?

- Everything is package-private.
- Everything is part of the same (unnamed) default package.

same (unnamed) default package

```
public class CeccoBeppe {
    void leap() {
        System.out.println("leapt");
    }
}
```

package private

```
public class Tane {
    public static void main(String[] args) {
        CeccoBeppe cb = new CeccoBeppe();
        cb.leap();
    }
}
```

Access Is Based Only on Static Types

```
package universe;
public interface BlackHole {
    void add(Object x);
}
```

add might look package private, actually public

```
package universe;
public class CreationUtils {
    public static BlackHole hirsute() {
        return new HasHair();
    }
}
```

```
package universe;
class HasHair implements BlackHole {
    Object[] items;
    public void add(Object o) { ... }
    public Object get(int k) { ... }
}
```

Try to predict if the compiler will allow each line of Client.

- Not part of the universe package!

```
import static CreationUtils.hirsute;
class Client {
 void demoAccess() {
  BlackHole b = hirsute();
  b.add("horse");
  b.get(0);
  HasHair hb = (HasHair) b;
 }
}
```

Yes
Yes
No
No

Can't refer to a HasHair blackhole from outside the package. However, can still called methods on objects of dynamic type HasHair (if we have a reference with allowable static type).

datastructur.es

A More Practical Example of Static Type Access Control

```
public class ArrayMap<K, V> implements Iterable<K> {
    private class KeyIterator implements Iterator<K> {
        private int wizardPosition;
        public KeyIterator () {
            wizardPosition = 0;
        }
    ...
    public Iterator<K> iterator() { return new KeyIterator(); }
```

Can never get a MapWizard (it's private), but the following works just fine:

```
ArrayMap<Gorgon, Pug> am = new ArrayMap<Gorgon, Pug>();
Iterator<Gorgon> it = am.iterator();
it.hasNext();
```

An earlier version of this slide erroneously said am.hasNext()

datastructur.es

Access Control at the Top Level

So far we've discussed how to control access to members.

- Also possible to control access at the top level (i.e. an entire interface or class).

Two levels:

- public
- no modifier: package-private

Determines who can see the existence of the class.

- Choices: Entire world vs. just members of same package.

`Cyberbrain` visible only to members of robot package.

```
package robot;

class Cyberbrain {
    ...
}
```

No such thing as a sub-package, BTW. ug.joshh.Animal and ug.joshh.Plant are two completely different packages.

datastructur.es

Purpose of the Access Modifiers

Access Levels:

- **Private** *declarations* are parts of the implementation of a class that only that class needs.
- *Package-private* declarations are parts of the implementation of a package that other members of the package will need to complete the implementation.
- ***Protected*** declarations are things that subtypes might need, but subtype clients will not.
- ***Public*** declarations are declarations of the specification for the package, i.e. what *clients* of the package can rely on. Once deployed, these should not change.

datastructur.es

Object Methods: Equals and toString()

datastructur.es

Objects

All classes are hyponyms of Object.

- **String toString()**
- **boolean equals(Object obj)**
- Class<?> getClass()
- int hashCode()
- protected Object clone()
- protected void finalize()
- void notify()
- void notifyAll()
- void wait()
- void wait(long timeout)
- void wait(long timeout, int nanos)

Won't discuss or use in 61B.

datastructur.es

toString()

If you want a custom String representation of an object, create a toString() method.

- Nothing particularly interesting about it, except that you can rely on it to generate a (nice?) String representation.

```
public String toString() {
    String mood;
    if (angry == true) {
       mood = "displeased";
    } else {
       mood = "happy";
    }
    return name + " is a " + mood + " " + breed +
                " weighing " + size + " standard lb units.";
}
```

```
Dog d = new Dog("Lucy",
                "Dachshund", 25);
System.out.println(d);
```

datastructur.es

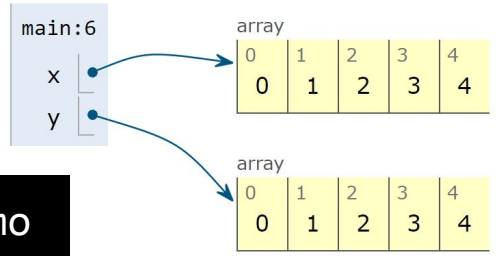
Equals vs. ==

As mentioned in an offhand manner previously, == and .equals() behave differently.

== checks that two variables reference the same object (compares bits in boxes):

```
public static void main(String[] args) {
    int[] x = new int[]{0, 1, 2, 3, 4};
    int[] y = new int[]{0, 1, 2, 3, 4};
    System.out.println(x == y);
}
```

```
$ java EqualsDemo
False
```



To test equality in the sense we usually mean it, use:

- `Arrays.equal` or `Arrays.deepEquals` for arrays.
- `.equals` for classes. Requires writing a `.equals` method for your classes.
  - Default implementation of `.equals` uses == (probably not what you want).

datastructure.es

Equals Method for Date

Rules for equals() are simple, but implementation is trickier than you might expect.

Webcast viewers: Try it out yourself. See study guide for starter code (Date.java).

```
public class Date {
  private final int month;
  private final int day;
  private final int year;

  public Date(int m, int d, int y) {
    month = m; day = d; year = y;
  }
```

datastructur.es

Equals Method for Date

Rules for equals() are simple, but implementation is trickier than you might expect.

One common approach at right.

```
public class Date {
  private final int month;
  private final int day;
  private final int year;

  public Date(int m, int d, int y) {
    month = m; day = d; year = y;
  }
```

```
public boolean equals(Object x) {
    if (this == x) return true;
    if (x == null) return false;
    if (this.getClass() != x.getClass()) {
       return false;
    }
    Date that = (Date) x;
    if (this.day != that.day) {
       return false;
    }
    if (this.month != that.month) {
       return false;
    }
    if (this.year != that.year) {
       return false;
    }
    return true;
}
```

CC BY NC SA

Rules for Equals in Java

Java convention is that equals must be an equivalence relation:

- Reflexive: x.equals(x) is true.
- Symmetric: x.equals(y) is true iff y.equals(x)
- Transitive: x.equals(y) and y.equals(z) implies x.equals(z).

Must also:

- Take an Object argument.
- Be consistent: If x.equals(y), then x must continue to equal y as long as neither changes.
- Never true for null, i.e. x.equals(null) must be false.

datastructur.es

The Future

There's plenty of Java we haven't covered, but this is enough.

- We'll expect you to start finding the syntax and built-in libraries you need, with tips in the HW/Projects for the trickier stuff. Don't use fancy external libraries like Apache Commons!

Weeks 7-14:

- Lectures: Data Structures and Algorithms
- Week 7: Project 2: "Advanced Programming"
- Weeks 8-14:
  - Labs: Implement data structures/algorithms.
  - HWs: Apply data structures algorithms.
  - Project 3: Efficiency oriented project.
- Optional textbook: Algorithms, 4th Edition

datastructur.es

Citations

First hit on google images from Exception:

http://education.oge.gov/training/module_files/ogewrkctr_wbt_07/exception.jpg

datastructur.es