CS61B Week 6 Note

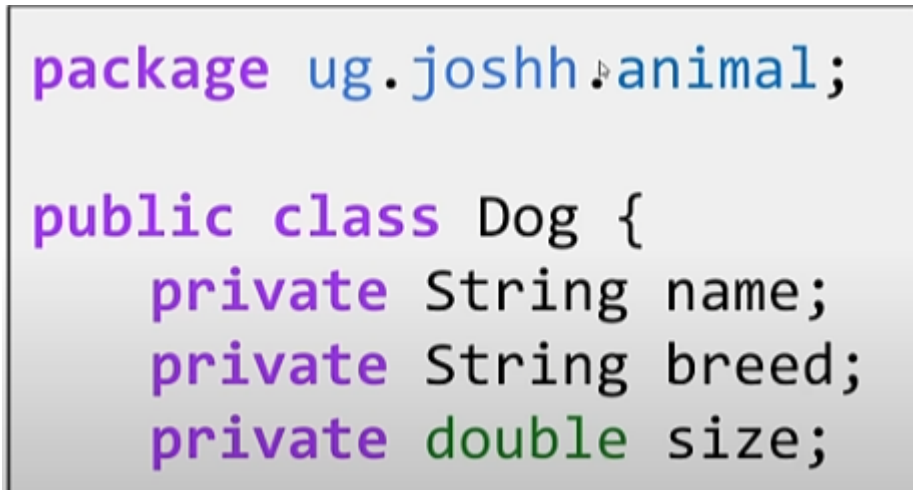
Gary Agasa

I.Packages, Access Control, Objects

One.Package

To address the fact that **classes might share names**:

- a package is a namespace that organizes classes and interfaces



```
package ug.joshh.animal;

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

- Naming convention:Package name starts with webisite address(backwards),e.g. org.junit refers to junit.org

> In this class ,we won't follow this convention.

1. creating a package:
   TWO STEPS:
   i. At the top of every file in the package,put the package name.
   ii. Make sure the file is stored in a folder with the appropriate folder name.
   > e.g. For a folder named ug.joshh.animal,use the folder ug/joshh/animal.

2. Using packages:
   - To use a class from package A in a class from package B,we use **canonical name** or use the **import** statement and use the simple 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(...);
```

- **the Default Package**:
  Any Java Class without a package name at the top are part of the "default" package.you should avoid using the default packade except for a very small example program.**You can't import the code from default package**

Two.Jar File

Jar files are just zipped files

- They don't keep your code safe!They are easy to unzip and transform back into java files.

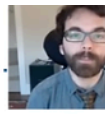
Three.Access Control

1. private
   - A subclass can't access a private member of its superclass
   - A class X in a package can't access the private member of its package buddy(同伴) Y
2. protected
   - **Protected** modifier allows package-buddies and subclasses to use a class member(i.e field/method/constructor)
3. package private
   - The way to get an access control level of package private is **not to put any key words**

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

The package members are considered less "secret" than subclass members

Purpose of the Access Modifiers

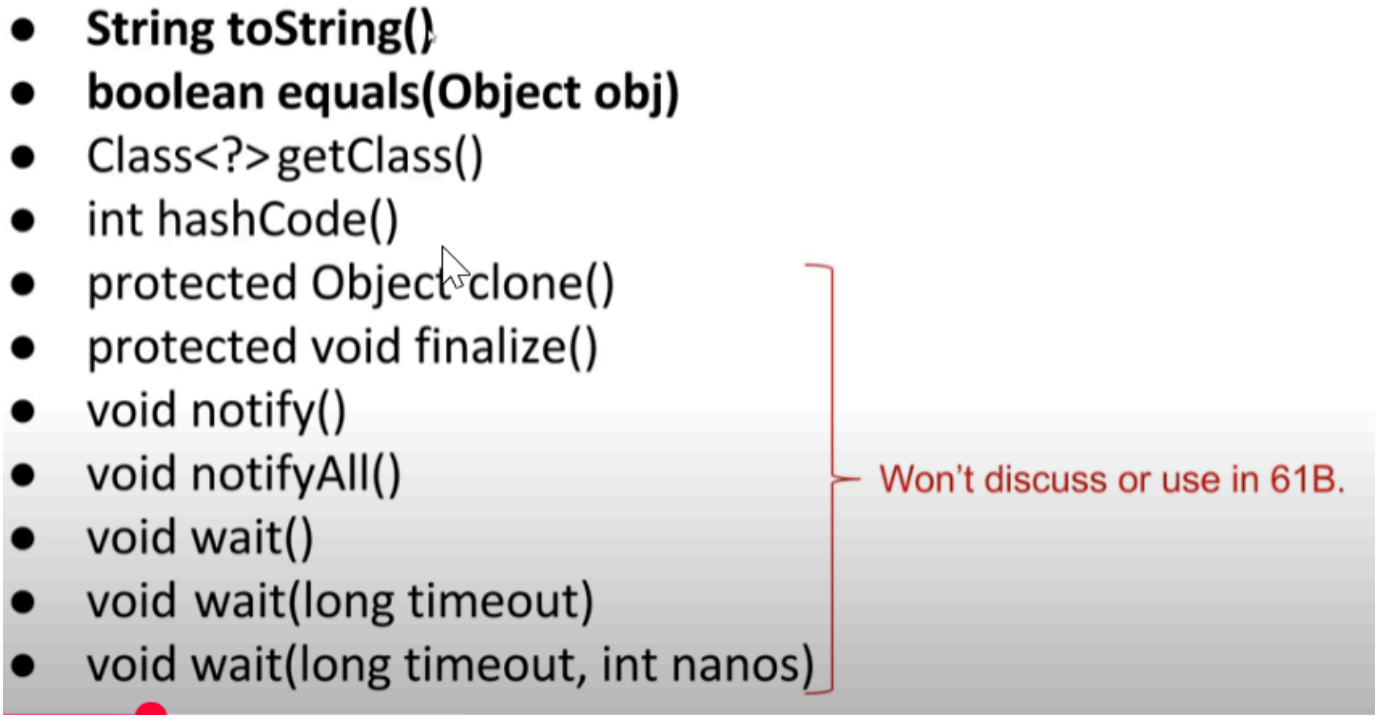


Access Levels:

- **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.

Four.Object methods

- All classes are hyponums of Objects,so they have some common functions:



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

```
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);/*false*/
}
```

   i. "=="checks that two variables **refer to the same object**.

   ii. .equals()

- **Array.equals()** or **Array.deepEquals()** can test the equality in the sense we usually usually mean it
- **.equals() for classes**:
  You can write a *.equals* method for your classes.
  The default implementation of *.equals()* is ==

```
package objectMethods;

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;
  }

  @Override
  /**When override the classes ,always put the @Override*/
  public boolean equals(Object o) {
     if (o == null) {
        return false;
     }
     if (this.getClass() != o.getClass()) {
        return false;
     }
     Date uddaDate = (Date) o;

     if (month != uddaDate.month) {
        return false;
     }
     if (day != uddaDate.day) {
        return false;
     }
     if (year != uddaDate.year) {
        return false;
     }
     return true;
  }
}
```

iii. 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.