Object

Hierarchy base class, toString( ), equals( ), hashCode( )

Ancestor of all classes

- In the spirit of OOP, every instance of a class is an object.
- Every Java class is part of the *Java class hierarchy*. The root of that hierarchy is **Object**.

  If you define a class like this:

  ```
  public class MyClass { …
  ```

  then `MyClass` is a *child* of `Object`.

- `Object` is defined in the `java.lang` package (from the Java API).

Object structure

- `Object` is not abstract – you can instantiate it.
- `Object` has no public, documented data members.
- Every class inherits `Object` methods. We discuss 3 methods in detail:
  - `public String toString()` – a String representation of the object.
  - `public boolean equals(Object x)` – true if this object “equals” the parameter x.
  - `public int hashCode()` – the hash value for this object. Distinct objects should have distinct integers.
- Many classes override the definitions of these methods.

toString( )

When you print an object:

```
   System.out.println(xx);
```

what is printed depends on how `xx`'s class defined `toString()`. `Object` defines it as the name of the class followed by the hash code of `xx`.

The `String` class overrides `toString()` to return the string of characters for `this`. So:

`“abcd”.toString()` returns `“abcd”`

If you don't override toString( )

... `toString()` returns the class name and the object's hash code

```
package edu.ncsu.csc216.samples.object;

public class StringExample {
   private int num;

   public static void main(String[] args) {
      StringExample x = new StringExample();
      System.out.println(x);
      Object o = new Object();
      System.out.println(o);
   }
}
```

Output:

```
edu.ncsu.csc216.samples.object.StringExample@677327b6
java.lang.Object@14ae5a5
```

and if you don’t override equals

…. equals( ) returns true only if the two references point to the same object (addresses of objects match).

```
package edu.ncsu.csc216.samples.object;

public class EqualExample {
    private int num = 50;

    public static void main(String[] args) {
        EqualExample x = new EqualExample();
        EqualExample y = new EqualExample();
        System.out.println ("x equals y?? " + x.equals(y));
    }
}
```

Output:

```
x equals y?? false
```

Overriding toString( )

- Called whenever a String is concatenated with an object.
- If you don't override toString(), the default return String is the form:

```
FullyQualifiedClassName@hashCode
```

- Example:

```
edu.ncsu.csc216.samples.Account@677327b6
```

- For a useful String return value, override toString() to reveal some of the fields.

```
public class Patient {
   private String name;
   private int age;

   @Override
   public String toString() {
      return "Patient [name=" + name
                  + ", age=" + age + "]";
   }
```

Overriding equals( )

Things to consider:

1. The default of equals() is the same as ==, which compares two objects to see if they are the *same* object.
2. Decide what equality should mean for your class. (Focus first on instance variables that are primitive.)
3. equals() has an Object type parameter, which will require a cast when you override.
4. For inheritance hierarchies, override equals() for each class in the hierarchy. Check for super.equals() in each but the top (not Object).
5. If you override equals(), then override hashCode() too.

Overriding equals( ) example

```
public class Patient {
   private String name;
   private int age;

   @Override
   public boolean equals(Object obj) {
      if (this == obj) {return true;}
      if (obj == null) {return false;}
      if (!(obj instanceof Patient)) {return false;}
      Patient other = (Patient) obj;
      if (age != other.age) { return false; }
      if (name == null) {
         if (other.name != null) { return false;}
         } else if (!name.equals(other.name)) {
            return false;
         }
      return true;
   }
```

Overriding hashCode( )

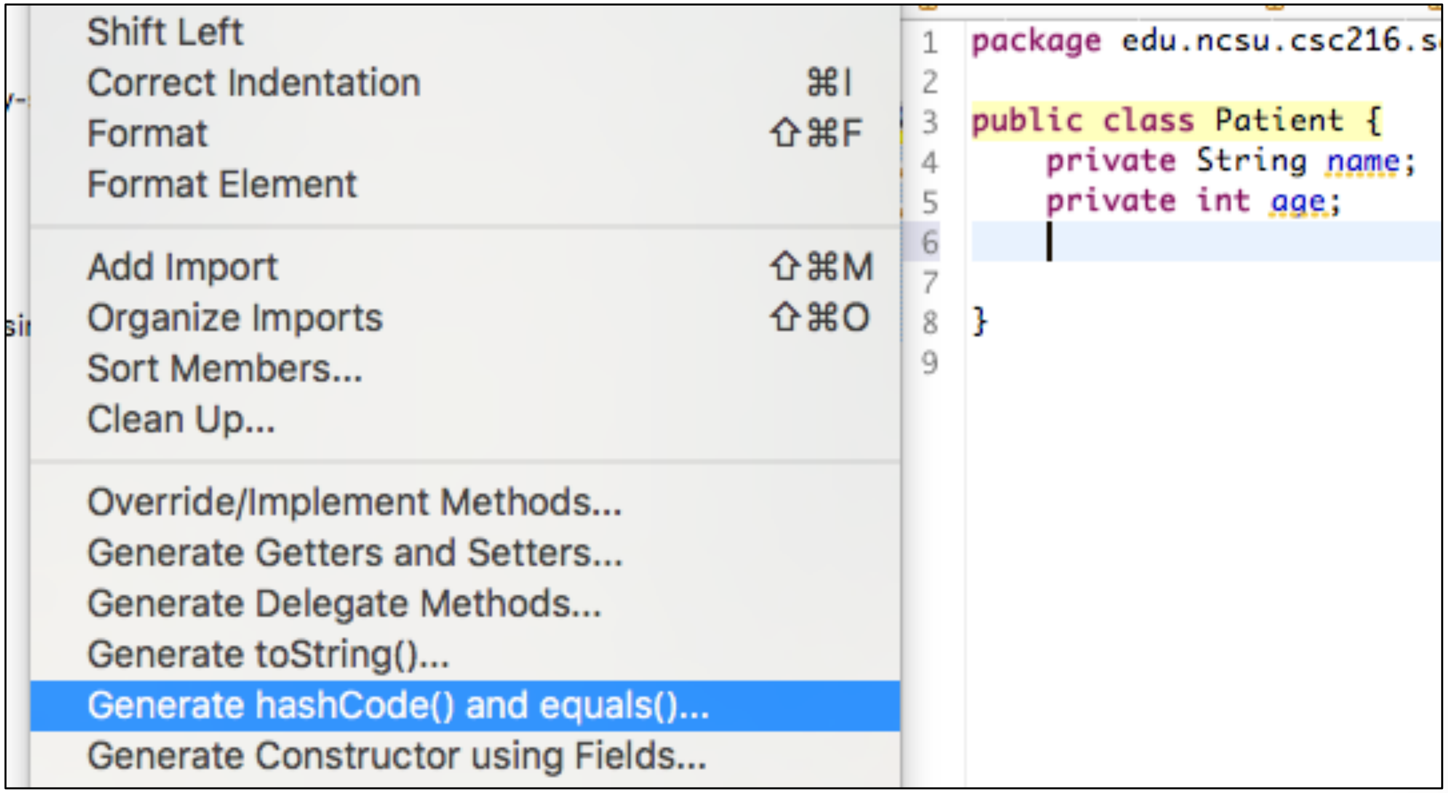
- hashCode() returns an unsigned integer value that is *likely* to be unique for different objects.
- Hash functions scramble info about the instance fields, typically using prime numbers.
- Objects that are the same should hash to the same value. Use the same fields to compare for equality and create the hash code
- If you override `equals()`, you must override `hashCode()`

```
public class Patient {
   private String name;
   private int age;

   @Override
   public int hashCode() {
      final int prime = 31;
      int result = 1;
      result = prime * result + age;
      result = prime * result
              + ((name == null) ? 0 : name.hashCode());
      return result;
   }
```

Overriding made easy

IDEs such as Eclipse make overriding easy. You can simply tell which Object method to override and Eclipse fills in as much code as it sees reasonable.

Eclipse. Select Source > Generate hashCode() and equals()

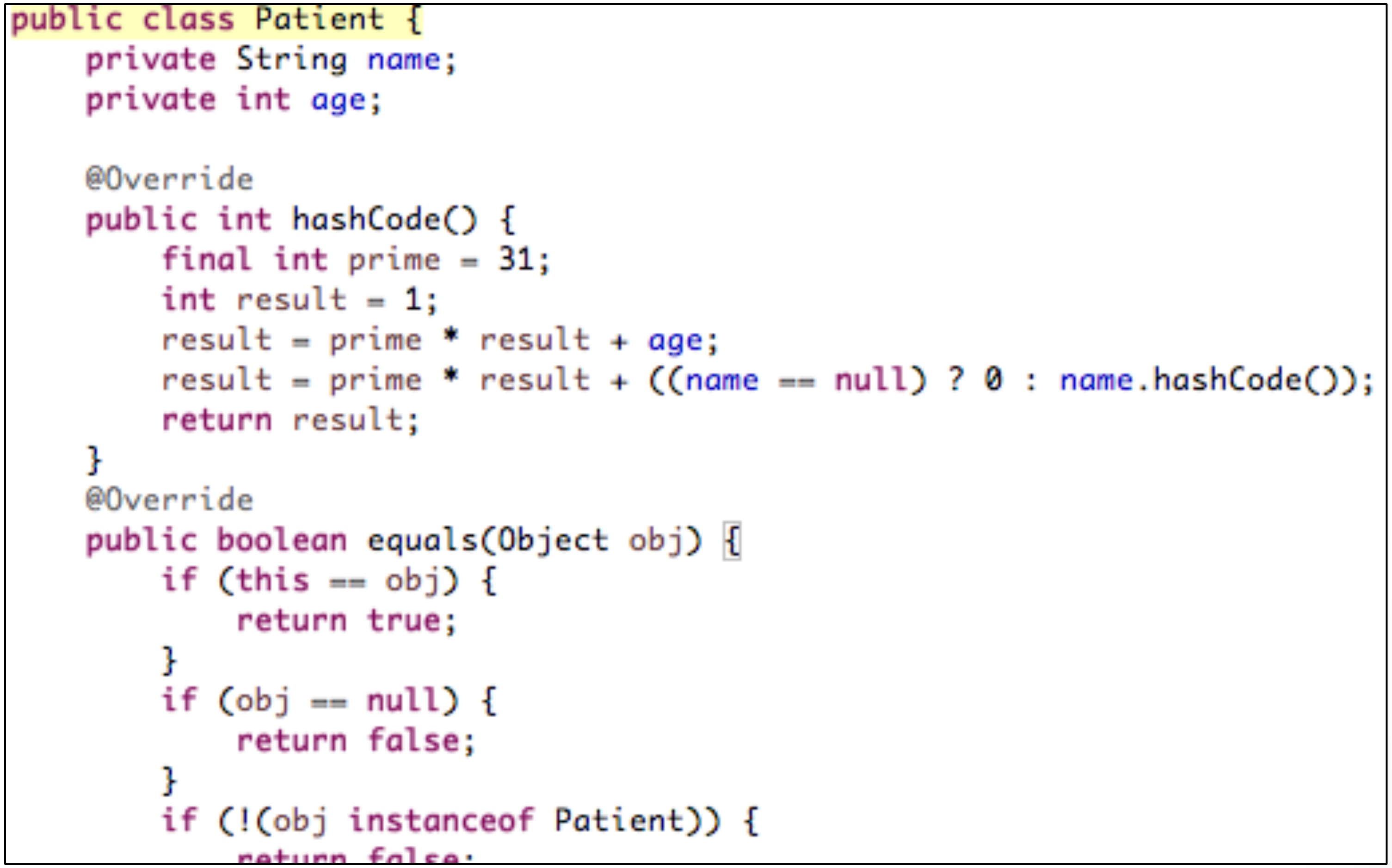


Shift Left
Correct Indentation ⌘I
Format ⇧⌘F
Format Element
Add Import ⇧⌘M
Organize Imports ⇧⌘O
Sort Members...
Clean Up...
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Generate toString()...
Generate hashCode() and equals()...
Generate Constructor using Fields...

```
package edu.ncsu.csc216.s

public class Patient {
    private String name;
    private int age;


}

```

Auto-generated equals(), hashCode()

Eclipse result of auto generation. You can edit to customize.



```
public class Patient {
    private String name;
    private int age;

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + age;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (!(obj instanceof Patient)) {
            return false;
```

Other Object methods

`Object` has additional methods, including:

- `protected Object clone() throws CloneNotSupportedException` -- a bitwise copy of this object.
- `void finalize()` – called by Java's garbage collector when there are no objects of this type remaining.
- `protected Object clone() throws CloneNotSupportedException` -- a bitwise copy of this object.
- `Class<?> getClass()` – the current/runtime class for this object.
- `void notify/notifyAll()` – wakes up threads waiting for this object's monitor.
- `void wait()` – 3 methods. Causes current thread to wait until another thread notifies this object.

Details on `Object` and all if its methods are in the Java API specifications:

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

The java version is part of the url (in this example, version 8).