CSE 687

Object Oriented Design

Nadeem Ghani
nghani@syr.edu
Office: CST 4-232
Office Hours: Friday 10 AM - 12 PM

TAs
Rui Zuo: rzuo02@syr.edu
Tuesday 10 - 11 AM

- HW 1 due Friday Sep 6th
- Quiz 1 Monday Sep 9th !

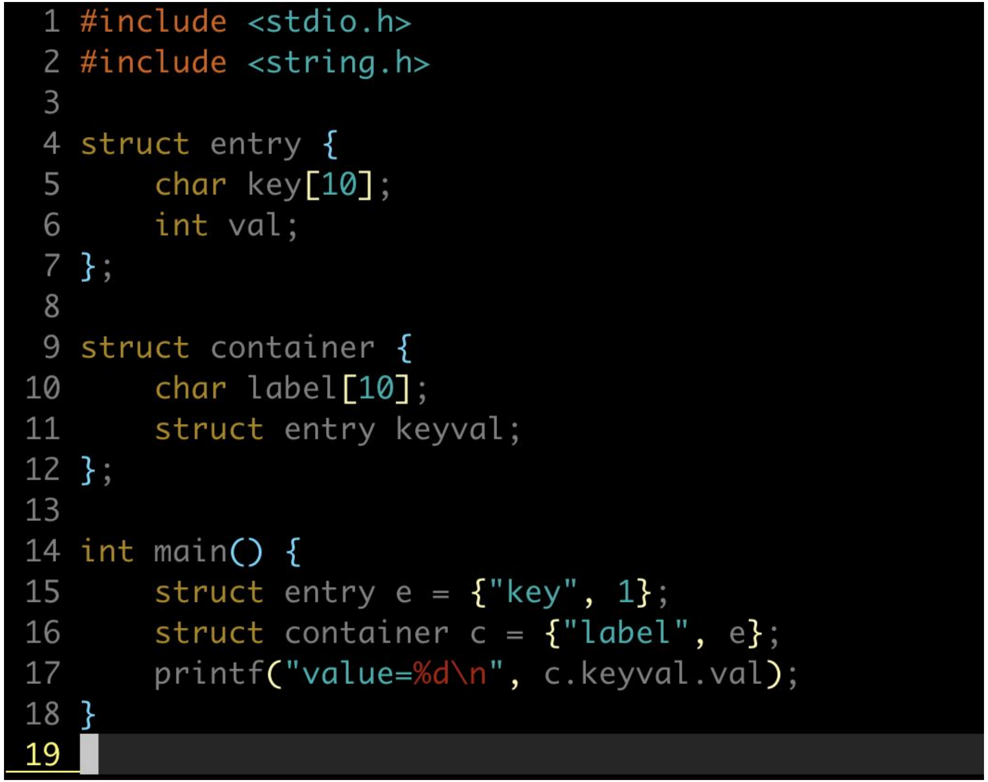
CSE 687

Extra Credit

CSE 687

Inheritance

- What is this code doing?



```c
#include <stdio.h>
#include <string.h>

struct entry {
    char key[10];
    int val;
};

struct container {
    char label[10];
    struct entry keyval;
};

int main() {
    struct entry e = {"key", 1};
    struct container c = {"label", e};
    printf("value=%d\n", c.keyval.val);
}
```

CSE 687

Inheritance

- The most direct translation of C code from previous slide to java..
- But in java, where was the parent object created?
- Should we be able to assign to `c.e.key`?

```
class Entry {
    int val;
    String key;
}
public class Container {
    String label;
    Entry e;
    public static void main(String[] args) {
        Container c = new Container();
        c.e.key = “key”;
    }
}
```

CSE 687

Inheritance

- We should be able to assign to `c.e.key`, provided c.e has been initialized!

```
class Entry {
    int val;
    String key;
}
public class Container {
    String label;
    Entry e;
    public static void main(String[] args) {
        Container c = new Container();
        c.e = new Entry();
        c.e.key = “key”;
    }
}
```

CSE 687

Inheritance

- But here’s a slightly different translation of the original C code into java
- Why can we access `c.key`?

```
class Entry {
     int val;
     String key;
}

public class Container extends Entry {
     String label;

     public static void main(String[] args) {
          Container c = new Container();
          c.key = “key”;
     }
}
```

CSE 687

Inheritance

- Parent - Child demo
  - super() from constructor
  - super.foo()
  - this.foo()
  - super.field
  - this.field
- Parent defines getSecret(), Child extends Parent, overrides getSecret()
  - Parent p = new Child()
  - p.getSecret(); // calls overridden getSecret()

CSE 687

Initialization

- initializing fields
    - at declaration
    - in constructor
    - initializer block
    - static initializer block
-
- What should you initialize a reference type to?
    - null vs “” or empty list
-
- local variables aren’t initialized!!

CSE 687

More Java

- Most Common Error: what does == do?

```
int first = 1;
int second = 1;
first == second; // true

String first = “hello”;
String second = “hello”;
first == second // true, but a recently added hack!
first.equals(second); // true
```

- ref equality!!

CSE 687

Low-Level Details

- primitive types vs boxed primitives

```
public class Unbelievable {
  static Integer i;

  public static void main(String[] args) {
    if (i==42){  // throws NullPointerException
      System.out.println(“Unbelievable”);
  }
}
```

- initialization to zero values; null for ref types
- prefer primitives to boxed values!

CSE 687

Object

- The root of Java class hierarchy
    - The only class that has no superclass

```
public class Object {
    public Object(){}
    public boolean equals(Object o) {
        return (this == obj);
    }
    public native int hashCode();
    public String toString() {
        return getClass().getName()+”@”+ Integer.toHexString(hashCode());
    }
```

- demo simple container

CSE 687

Object

- The root of Java class hierarchy
  - The only class that has no superclass

```
public class Object {
    public Object(){}  // constructor
    public boolean equals(Object o) {
        return (this == obj);
    }
    public native int hashCode();
    public String toString() {
        return getClass().getName()+”@”+ Integer.toHexString(hashCode());
    }
```

CSE 687

Object

- The root of Java class hierarchy
  - The only class that has no superclass

```
public class Object {
    public Object(){}
    public boolean equals(Object o) { // compares ref equality
        return (this == obj);
    }
    public native int hashCode();
    public String toString() {
        return getClass().getName()+”@”+ Integer.toHexString(hashCode());
    }
```

CSE 687

Object.equals()

- Default behavior compares ref equality

```
Integer n = new Integer(12345);
Integer alsoN = new Integer(12345);
System.out.println(n == alsoN); // false
System.out.println(n.equals(alsoN)); // true
```

- Integer provides its own equals method

```
public boolean equals(Object obj) {
    if (obj instanceOf Integer) {
        return value == ((Integer)obj).intValue();
    }
    return false;
}
```

CSE 687

Object.equals()

- Contract?!
    - reflexive `x.equals(x) == true`
    - symmetric `x.equals(y) == y.equals(x)`
    - transitive `x.equals(y) == true && y.equals(z) == true implies x.equals(z) == true`
    - consistent repeated calls give same result
    - null check `x.equals(null) == false`

- Not easy to break with simple classes
- But not too difficult to break when inheritance is involved
- Can be a time sink to find if broken

- Test thoroughly!

CSE 687

Object.equals() recipe

```
public boolean equals(Object o) {
    if (o == this) return true;
    if (o==null || !(o instanceOf [class])) {
        return false;
    }
    [class] that = ([class]) o;
    // now compare all significant fields
    // use equals() for reference types comparison
    // use == for primitive types
    if (that.f1.equals(this.f1) && that.int == this.int) {
        return true;
    }
    return false;
```

CSE 687

Object

- The root of Java class hierarchy
    - The only class that has no superclass

```
public class Object {
    public Object(){}
    public boolean equals(Object o) {
        return (this == obj);
    }
    public native int hashCode();
    public String toString() {
        return getClass().getName()+”@”+ Integer.toHexString(hashCode());
    }
```

CSE 687

Object

- Think of equals() and hashCode() as a pair
- Decide if your class needs an equals method
- If yes, also provide hashCode method

```
public class Object {
    public Object(){}
    public boolean equals(Object o) { // compares ref equality
        return (this == obj);
    }
    public native int hashCode();
    public String toString() {
        return getClass().getName()+”@”+ Integer.toHexString(hashCode());
    }
```

CSE 687

Object

- Decide if your class needs an equals method. If yes, also provide hashCode().
- Assuming Entry.equals() as shown here…

```
class Entry {
   int val;
   String key;
   public Entry(String k, int v) { key = k; val = v;}
   @Override
   public boolean equals(Object o) {
       if (o == this) return true;
       if (o==null || !(o instanceof Entry)) return false;

       Entry that = (Entry) o;
       return this.key.equals(that.key) && this.val == that.val;
   }
}
```

CSE 687

Object

- … java.util classes give unexpected results!

```
Entry e1 = new Entry("k1", 1);
Entry e2 = new Entry("k1", 1);

HashSet<Entry> set = new HashSet<>();
set.add(e1);
System.out.println(set.contains(e2));  // prints false

HashMap<Entry, String> map = new HashMap<>();
map.put(e1, "one");
System.out.println(map.get(e2));  // prints null
```

CSE 687

Object.hashCode()

- Contract?!
  - consistent repeated calls give same result, if obj hasn’t changed
  - equality `x.equals(y) implies that x.hashCode()== y.hashCode()`
  - not equal better performance if `!x.equals(y) implies that x.hashCode() != y.hashCode()`

- Decent performance is easy

- Test thoroughly!
- Good to know: `System.identityHashCode()`

CSE 687

Object.hashCode() recipe

```
public int hashCode() {
    int result = 17; // 17 is arbitrary

    //for each field in this obj
        c = hashcode(field)
        result = 31 * result + c; // 31 is traditional; any prime

    return result
}
```

CSE 687

hashCode() example

```
// java.lang.StringUTF16

public static int hashCode(byte[] value) {
        int h = 0;
        int length = value.length >> 1;
        for (int i = 0; i < length; i++) {
            h = 31 * h + getChar(value, i);
        }
        return h;
}
```

CSE 687

Object

- Always override toString()

```
public class Object {
    public Object(){}
    public boolean equals(Object o) { // compares ref equality
        return (this == obj);
    }
    public native int hashCode();
    public String toString() {
        return getClass().getName()+"@"+ Integer.toHexString(hashCode());
    }
}

int[] data = new int[]{1, 2, 3};
System.out.println(data);  // prints [I@8efb846
```