## Lecture 5: Cloning, Copying, Methods Common to All Objects

Budd – Ch 12

1. inheritance “is-a” relationship creates complications with memory, on the stack, memory requirements must be determined statically at runtime but sometimes all the information about the variable isn’t known yet, so variables need to be declared on the heap which takes more memory
2. problem: subclasses introduce data not present in the superclass
3. options: (1) allocate space for base class only, (2) allocate max space for any value in the base or subclass, (3) allocate only the space for a pointer, allocate space for the needed base and subclass values at runtime on the heap, point the pointer to the heap
4. C++ has runtime allocated variables (dynamic variables)
5. shallow copy shares instance variables with the original
6. deep copy creates new copies of the instance variables, recursively copying tiered fields (like all the key pairs of a dictionary field)
7. cloning: Java Object class has a protected method called clone.

class PlayingCard implements Cloneable {

public Object clone() throws CloneNotSupportedException {

Object newCard = super.clone(); // shallow copy

// write deep copy code here …

return newCar;

}

}

1. identity versus equality
   1. identity: are two objects precisely the name object, being viewed at a different angle? (only 1 object exists, same memory address)
   2. equality: same fields, size, and shape, different memory address, domain-specific

Bloch Item 10-14

Item 10: Obey the general contract when overriding equals

Don’t when: each instance of class is unique, equality won’t be tested, subclasses’ superclass already has an equals that works for that class, class is private or package private, class is a value class like an ENUM.

Do when: has logical equality and superclass doesn’t have a equals method that works already.

Object’s equals contract

1. reflexive, an object is always equal to itself!
2. symmetric, for two equal objects, they both need to return true when taken from left
3. transitive, if one object equals a second, and the second equals a third, the first object also equals the third (favor composition over inheritence to avoid issues with this)
4. consistent, if two objects are equal, they need to remain equal unless one is modified
5. equals on null is always false

Steps to Override equals

1. use @Override in fxn def
2. Use the == operator to check if the argument is a reference to *this* object, if so, return false
3. Use the instanceof operator to check if the argument has the correct type
4. Cast the argument to the correct type
5. For each significant field in the class, check the fields match

Text

Description automatically generated

Item 11: If you overrode equals you must override hashCode

Override hashCode when you overrode equals. EQUAL OBJECTS HAVE EQUAL HASHCODES.

Text

Description automatically generated

Item 12: Always override toString()

Self explanatory

Item 13: Override clone() (in Java) judiciously

clone() returns a field-by-field copy of the object, or throws a CloneNotSupportedException if method is not public (don’t throw exception for public, try/catch instead).

clone() does not work on final fields.

Clone()’s contract

1. x.clone() != x // returns true, identify check (different addresses)
2. x.clone().getClass() == x.getClass // returns true, same class
3. x.clone().equals(x) // typically true
4. must not do any harm to the original object

Item 14: Consider implementing Comparable interface

compareTo() can be implemented from the Comparable interface in Java.

throws ClassCastException if the type of the compared object doesn’t match.

returns an int, not a boolean.