CSSE 220 Day 16

Object: the superest class of all
Inheritance and text in GUIs

Check out *CloneAndText* from SVN

Questions?

- Interfaces
- Inheritance
- extends *vs* implements
- abstract classes and methods
- polymorphism
- Hardy's taxi
- anything else

I, Object

- The superest class in Java

Object

- **Every** class in Java inherits from **`Object`**
  - Directly and **explicitly**:
    - `public class String extends Object {…}`
  - Directly and **implicitly**:
    - `class BankAccount {…}`
  - **Indirectly**:
    - `class SavingsAccount extends BankAccount {…}`

Q1

Object Provides Several Methods

- `String toString()`
- `boolean equals(Object otherObject)`
- `Class getClass()`
- `Object clone()`
- …

Often overridden

Often useful

Often dangerous!

Q2

Overriding toString()

- Return a concise, human-readable summary of the object state

- Very useful because it's called automatically:
  - During string concatenation
  - For printing
  - In the debugger

- `getClass().getName()` comes in handy here…

Overriding equals(Object o)

- Should return true when comparing two objects of same type with same "meaning"
  - Must check types—use **instanceof**
  - Must compare state—use **cast**
- Example: Similar to what did in Fraction:

```
@Override
public boolean equals(Object obj) {
    // First, check type of other object
  if (!(obj instanceof SafeDepositBox))
    return false;
    // Next, cast the other object so we can get at the fields
  SafeDepositBox otherBox = (SafeDepositBox) obj;
    // Finally, compare all instance fields using == for
    // primitives, equals method for objects.
  return this.boxNumber == otherBox.boxNumber;
```

The Reason for clone( )

- Avoiding representation exposure:
  - i.e. returning an object that lets other code change our object's state

```
public class Customer {
    private String name;
    private BankAccount acct;
    …
    public String getName() {
        return this.name;  // ← OK!
    }

    public BankAccount getAccount() {
        return this.acct; // ← Rep. exposure!
    }
```

Book says (controversiallly) to use
`return (BankAccount) this.acct.clone();`"

Q3,4

The Trouble with `clone()`

- `clone( )` is supposed to make a *deep copy*
  1. Copy the object
  2. Copy any mutable objects it points to
- `Object`'s `clone( )` handles 1 **but not 2**
- *Effective Java* includes a seven page description on overriding `clone()`:
  - "[You] are probably better off providing some alternative means of object copying or simply not providing the capability."

Effective Java, by Joshua Block

Q5,6

Alternatives to clone()

- Copy constructor in Customer:
  - `public Customer(Customer toBeCopied) {…}`
- Copy factory in BankAccount:
  - `public abstract BankAccount getCopy();`
- Fixed Example:

  -
    ```
    public BankAccount getAccount() {
        return this.acct.getCopy();
    }
    ```

Add method stub to BankAccount

- Note that doing this changes BankAccount into an abstract class:

```
/**
 * @return a deep copy of this account
 */
public abstract BankAccount getCopy();
```

Fix representation exposure:

```
public Customer(String name, BankAccount
account) {
  this.name = name;
  // TODO 6: fix representation exposure
  // this.account = account;
  this.account = account.getCopy();
}

public BankAccount getAccount() {
  // TODO 7: fix representation exposure
  // return this.account;
  return this.account.getCopy();
}
```

Add a copy constructor

```
/**
 * Constructs a deep copy of the given
 * customer object.
 *
 * @param toBeCopied
 */
public Customer(Customer toBeCopied) {
  this.name = toBeCopied.name;
  this.account = toBeCopied.account.getCopy();
}
```

Better Frames Through Inheritance

- GUI concepts are review.

  Some details are new.

  Such as how the inner class refers to instance fields of the enclosing class.

BallWorlds

- Demo
  UML diagram (correction!)
  Begin work (with Hardy partner)