CSSE 220 Day 16

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

Check out *Inheritance2* from SVN

Questions?

Project Team Preference Survey

- On ANGEL, under Lessons → Assignments
- Preferences help me to choose teams; I also consider your performance so far in the course
- Complete the survey by **Monday, Jan 28, 2012, noon**
- Most teams will have 3 students
- Are you willing to be on a team of 2?
- List up to 5 students you'd like to work with, highest preference first.
  - You may not get your first choices, so it's a good idea to list more than two
  - Best to choose partners whose commitment level and current Java coding/debugging ability is similar to yours
- List up to 2 students you'd prefer NOT to work with
  - I'll do my best to honor this, but I must find a team for everyone.

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

Sometimes 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…

Q3

Overriding *equals(Object o)*

- Should return true when comparing two objects of same type with same “meaning”
- How?
  - Must check types—use *instanceof*
  - Must compare state—use **cast**
- Example…

Q4

Polymorphism

- Review and Practice

Polymorphism and Subclasses

- A subclass instance **is a** superclass instance
  - Polymorphism still works!
  - ```
    BankAccount ba = new SavingsAccount();
    ba.deposit(100);
    ```
- But not the other way around!
  - ```
    SavingsAccount sa = new BankAccount();
    sa.addInterest();
    ```
- Why not?

Another Example

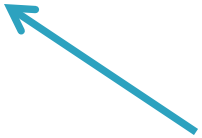
- Can use:
  - ```
    public void transfer(double amt, BankAccount o){
        this.withdraw(amount);
        o.deposit(amount);
    }
    ```
    in BankAccount
- To transfer between different accounts:
  - ```
    SavingsAccount sa = …;
    ```
  - ```
    CheckingAccount ca = …;
    ```
  - ```
    sa.transfer(100, ca);
    ```

Summary

- If B extends or implements A, we can write

```
A x = new B();
```

Declared type tells which methods x can access. Compile-time error if try to use method not in A.



The actual type tells which class' version of the method to use.

- Can cast to recover methods from B:

```
((B)x).foo()
```

Now we can access all of B's methods too.

If x isn't an instance of B, it gives a run-time error (class cast exception)

Q5-7, hand in when done, then start reading BallWorlds spec

BallWorlds

- Meet your partner (see link in part 3 of spec)
- Carefully read the requirements and provided code
- Ask questions (instructor and TAs).

BallWorlds Worktime

- Pulsar, Mover, etc.