CSSE 220

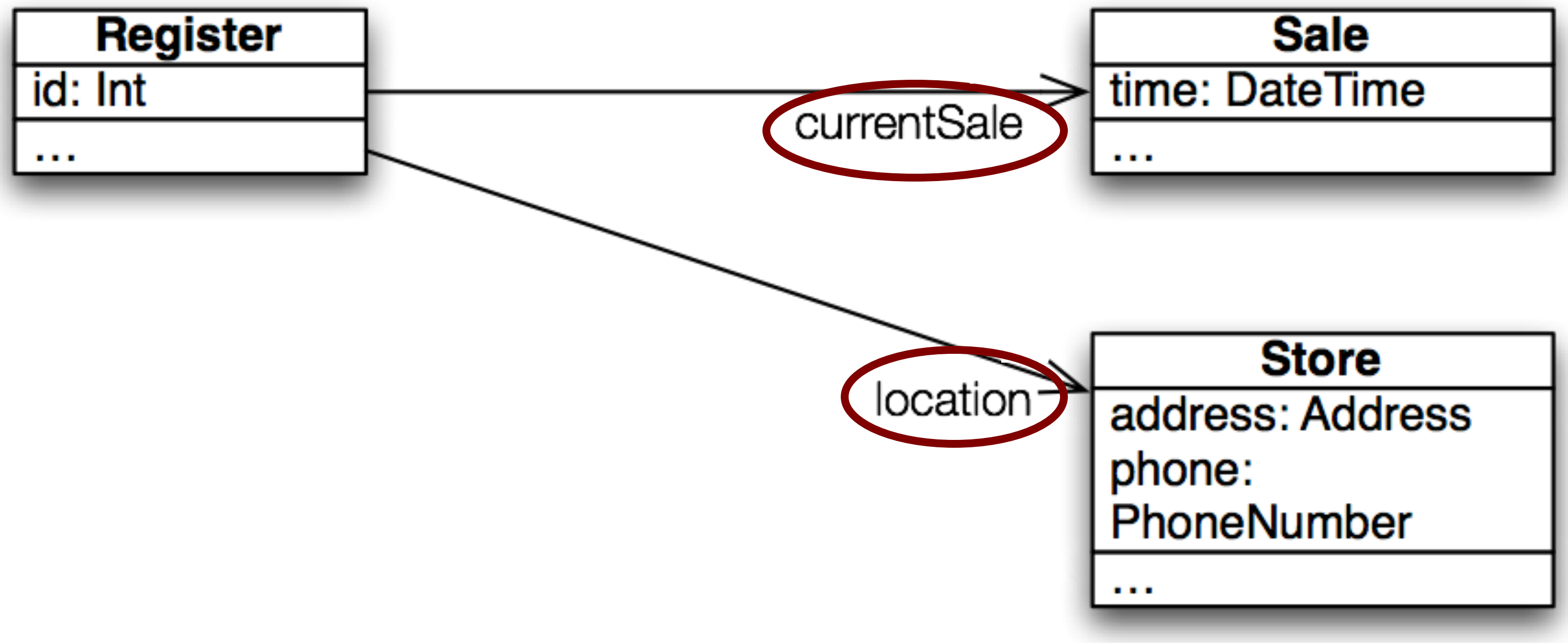
Object & Polymorphism

Check out *Polymorphism* from SVN

UML Class Diagram Review

- Inheritance, Associations, and Dependencies

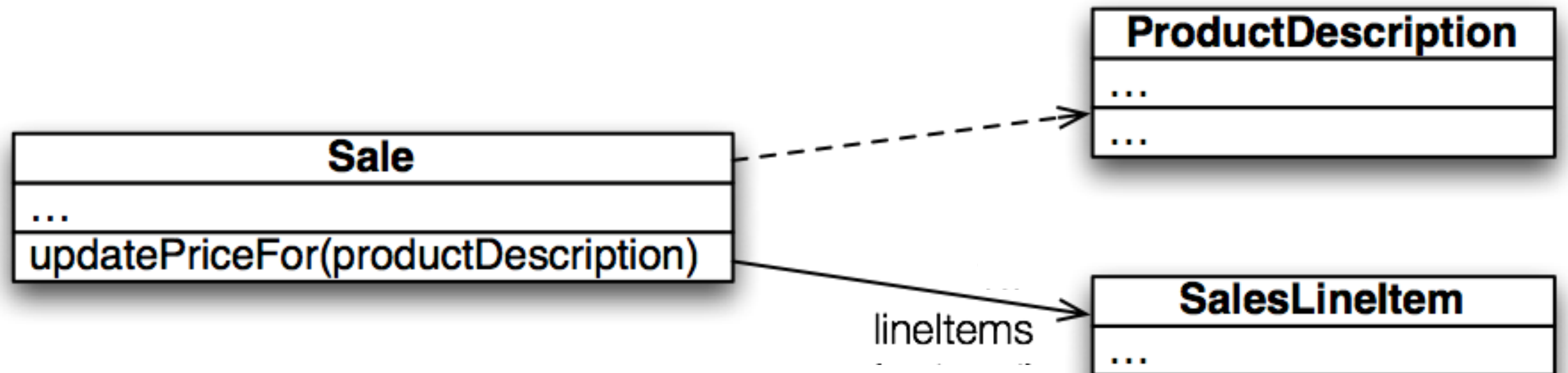
Recall UML: Associations



Solid line, open arrowhead = “has-a”

Recall UML: Dependencies

Dependency lines are dashed



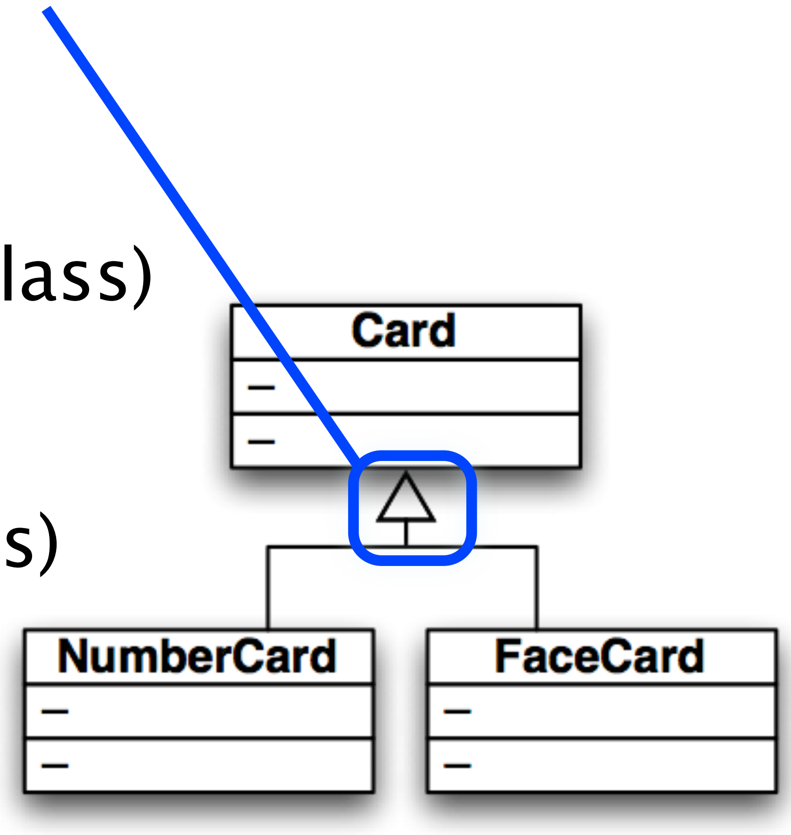
Field association lines are solid

Use association lines only when an item is stored as a field.

Two types of open arrowheads

Recall UML: Inheritance

- Generalization (superclass)
- Specialization (subclass)



Closed arrowhead = “is-a”.
Two types: solid line= inherits, dotted line = implements

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()` OR `getClass().getSimpleName()`** comes in handy here…

Q3

Overriding `equals(Object o)`

- `equals(Object foo)` – should return true when comparing two objects of same type with same “meaning”

- How?
  - Must check types—use **`instanceof`** **`OR`** **`getClass().isAssignableFrom(foo.getClass())`**
  - Must compare state—use **cast**

Recall casting a variable: Taking an Object of one particular type and “turning it into” another Object type

Q4

Polymorphism

- Review and Practice

Recall: 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?

BOOM!

Another Example

- Can use:

```
public void transfer(double amount, 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)

Determining Method at Runtime

- Step 1: Identify the Declared/Casted Type
  - This is the item to the left of the variable name when the variable was declared:
    - BankAccount sa = new SavingsAccount();

Declared Type

  - Declared Type may be changed due to a cast:
  - ((SavingsAccount)sa).addInterest();

Casted Type

  - If there is a casted type, record that, otherwise use the declared type.

Determining Method at Runtime

- Step 2: Identify the Instantiation/Actual Type
  - This is the type on the right hand side of the equal sign the last time the variable was assigned to:
    - BankAccount sa = new SavingsAccount();

Instantiation Type

  - Record the instantiation type

Determining Method at Runtime

- Step 3: Check for Compilation Errors

  Calling a method that is not available based on the declared or casted type of the object

  BankAccount sa = new SavingsAccount();

  sa.addInterest();

  Compiler Error: BankAccount does not have addInterest

  Incompatible type assignment

  SavingsAccount x = new BankAccount();

  Compiler Error: BankAccounts can not be stored in SavingAccount typed variables

  Invalid cast: casting to a type that isn’t in the tree below the declaration type.

  BankAccount sa = new SavingsAccount();

  ((SafetyDepositBox)sa).depositItem();

  SafetyDepositBox is not below BankAccount.

  Cannot instantiate interfaces or abstract classes!

Determining Method at Runtime

- Step 4: Check for Runtime Errors

  Runtime errors are caused by invalid casting.

  An item may only be cast to a type IF:

  - The instantiation type matches the casted type
  - The casted type is between the declaration type and the instantiation type

  ```
  BankAccount sa = new SavingsAccount();
  ((CheckingAccount)sa).deductFees();
  ```

  Runtime Error: SavingsAccount is not a CheckingAccount

  ```
  Account a = new CheckingAccount();
  ((BankAccount)a).deposit();
  ```

  This is valid because a CheckingAccount is a BankAccount

Determining Method at Runtime

- Step 5: Find Method to Run
  - Find the instantiation type in the hierarchy.
    1. If that type implements the given method, then use that implementation.
    2. Otherwise, move up to the parent type and see if there's an implementation there.
       a. If there is an implementation, use that.
       b. Otherwise, repeat step 2 until an implementation is found.

Exercise

- Do questions 5 through 7 from Quiz.
- Please hand them in when done and then start reading the BallWorlds specification on your schedule page.

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

BallWorlds

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

BallWorlds Teams posted on Moodle

Look over the BallWorlds UML Class Diagram and start Questions.

Check out *BallWorlds* from SVN

BallWorlds Worktime

- Pulsar, Mover, etc.