Lecture 09 Summary

- Mutability (from L08)
- Navigability
- Class Variables
- Class Methods

By the end of this lecture, you will be able to distinguish between *mutable* and *immutable* classes.

You will also be able to describe the *navigability* of an object model.

You will also be able to create *unidirectional* and *bidirectional* associations between objects.

You will also be able to create classes with *class variables* and *class methods*.

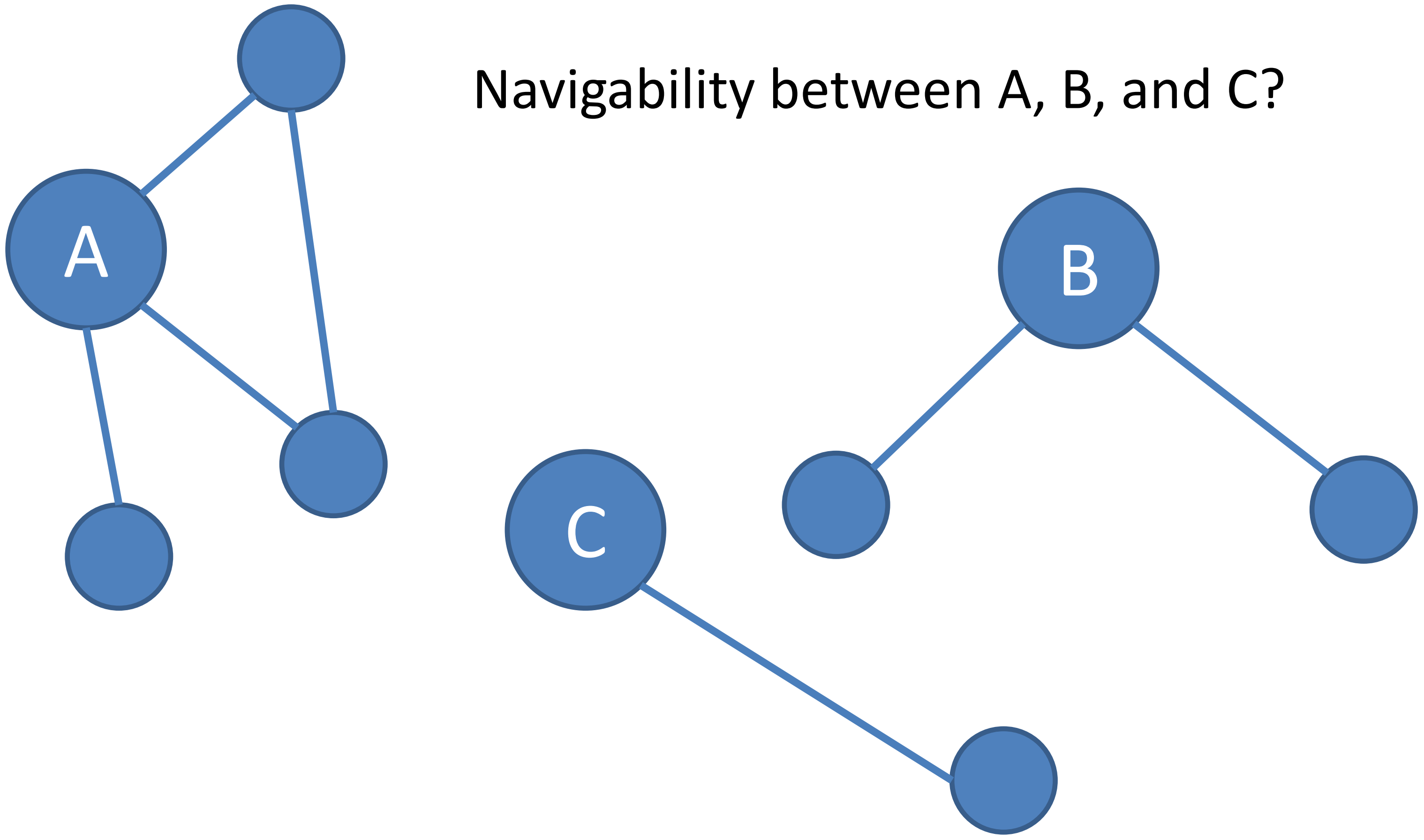
Mutability Summary

- Things to check:
  - Are all of the instance variables private?
  - Do any public methods change the instance variables?
  - Do any of the getters return a reference to a mutable instance variable?

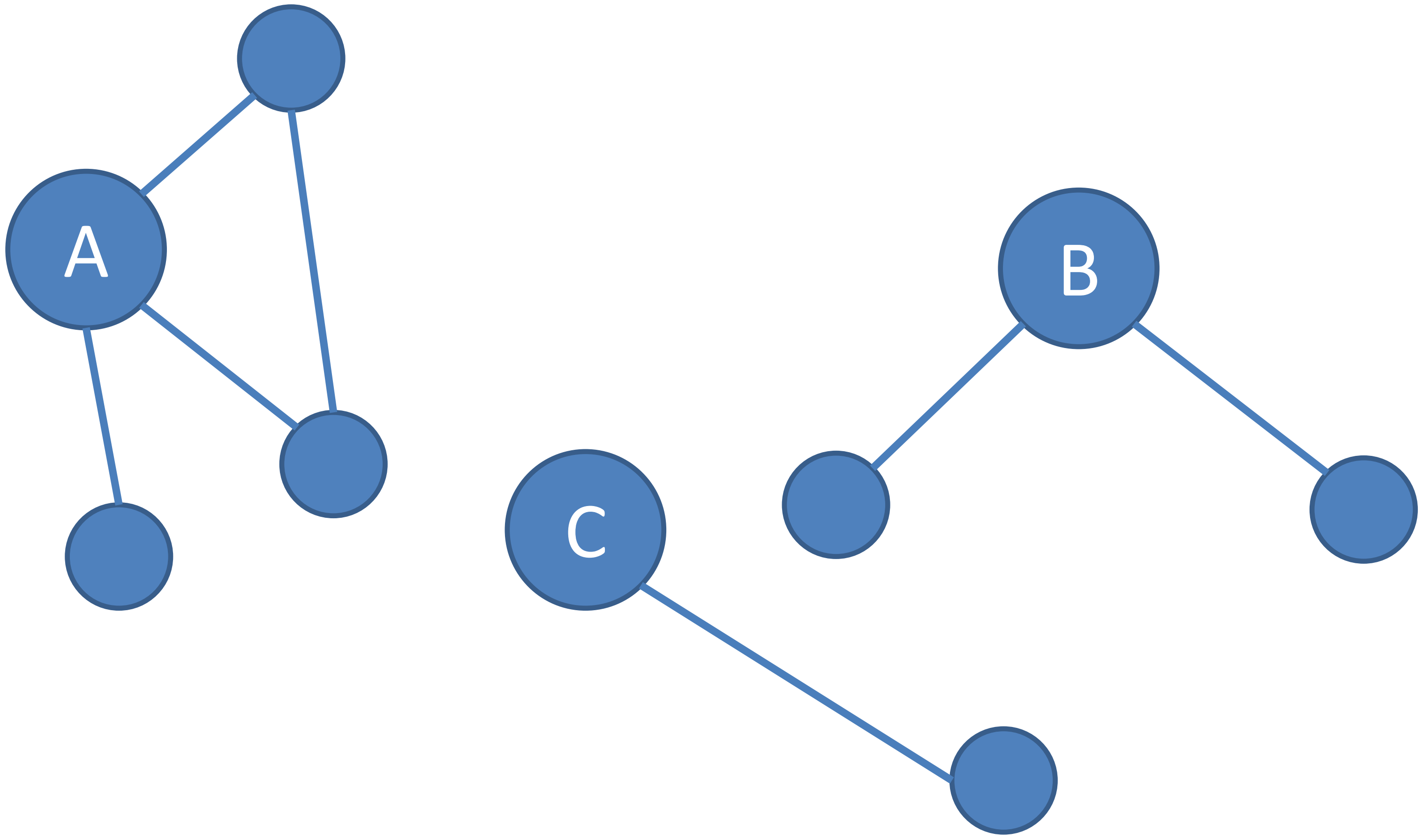
Navigability

Consider the object model

Navigability between A, B, and C?



Exercise



Navigability in OO Languages

- Object-oriented (OO) languages support *unidirectional* associations (e.g., has-a)

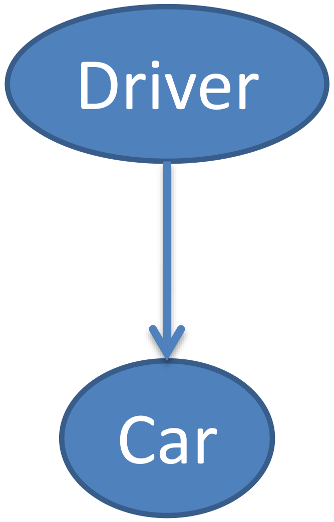
Example

Car.java:

```
public class Car
{
    ... attributes of car ...
}
```

Driver.java:

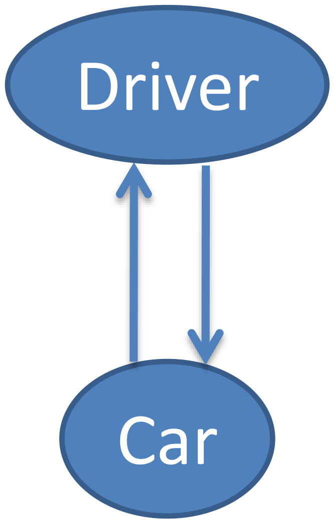
```
public class Driver
{
    private Car car;
    ... other attributes of driver ...

    public void setCar(Car aCar)
    {
        car = aCar;
    }
}
```



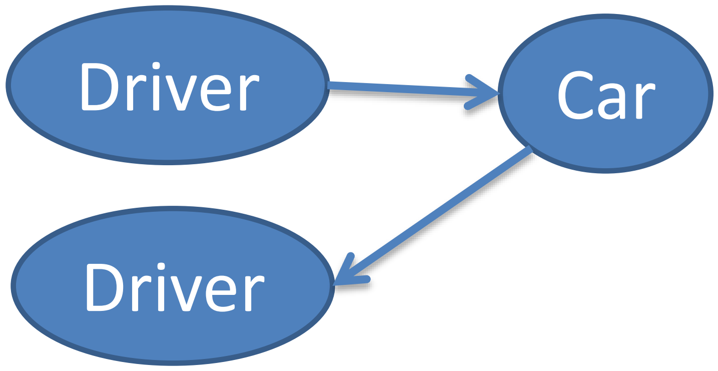
How could we achieve a bidirectional association between Driver and Car?

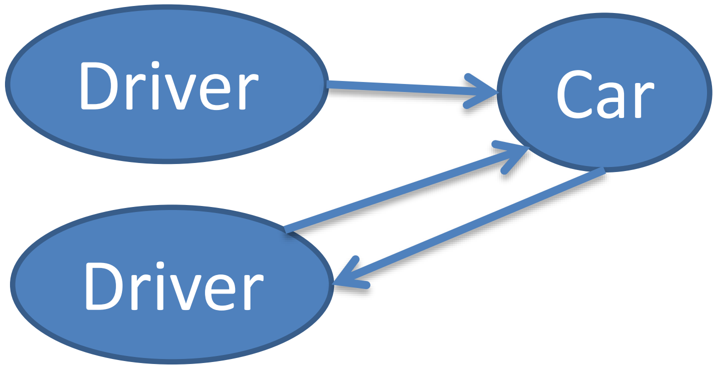
Possibilities

What we want



What we don't want





Draw the object model and describe the navigability...

Customer.java:
```
public class Customer
{
    private Transaction[] transactions;
}
```
BankAccount.java:
```
public class BankAccount
{
    private Customer customer;
}
```
Transaction.java:
```
public class Transaction
{
    private BankAccount bankAccount;
}
```

Analysis:

- What would these methods look like:
  - `addTransaction`,
  - `setBankAccount`, and
  - `setCustomer`?

- Describe a simpler design

Navigability Summary

- OO Languages only *directly* support unidirectional associations
- Bidirectional associations (that ensure consistency) require extra work
- Result: object model less navigable

- Design question: *which object is likely to require information from the other object?*

Class Variables and Class Methods

Example

```
public class SquareRootProgram
{
    public static void main(String[] args)
    {
        if (args.length != 1)
        {
            System.out.println("Too few arguments");
        }

        double arg = Double.parseDouble(args[0]);
        double root = Math.sqrt(arg);

        System.out.println("The square root is: " + root);
    }
}
```

Example

- What are “Math”, “Double”, and “System”?
- Do we have any instances of a “Math” object?
- http://java.sun.com/javase/6/docs/api/

Class Variables

- Variables that are associated with a particular *class*.
- Can think of as: variables that are the same for all instances.

Example

```
public class Customer
{
    private static int numInstances = 0;

    ... instance variables ...

    public Customer()
    {
        numInstances++;
        ... other initialization code ...
    }

    ... instance methods ...
}
```

Class Method

- Methods that are associated with a particular *class*.
- Can think of as: methods that do not require knowledge about any particular instance.

Example

```java
public class Customer
{
    private static int numInstances = 0;

    public Customer()
    {
        numInstances++;
    }

    public static int getNumInstances()
    {
        return numInstances;
    }
}
```

Example: Constants

```
public class Mole
{
    public static final float AVAGADRO_CONSTANT = 6.02E+23f;
    ...
}
```

Summary

- Class variables and class methods are associated with a particular *class*, but not to any particular *instance* of that class.
- The `static` keyword indicates a class variable or method.
- The `final` keyword indicates that something is unchangeable

Lecture 09 Summary

- Mutability
- Navigability
- Class Variables
- Class Methods

Next Class

- In-Class Coding Examples
- Midterm Preparation