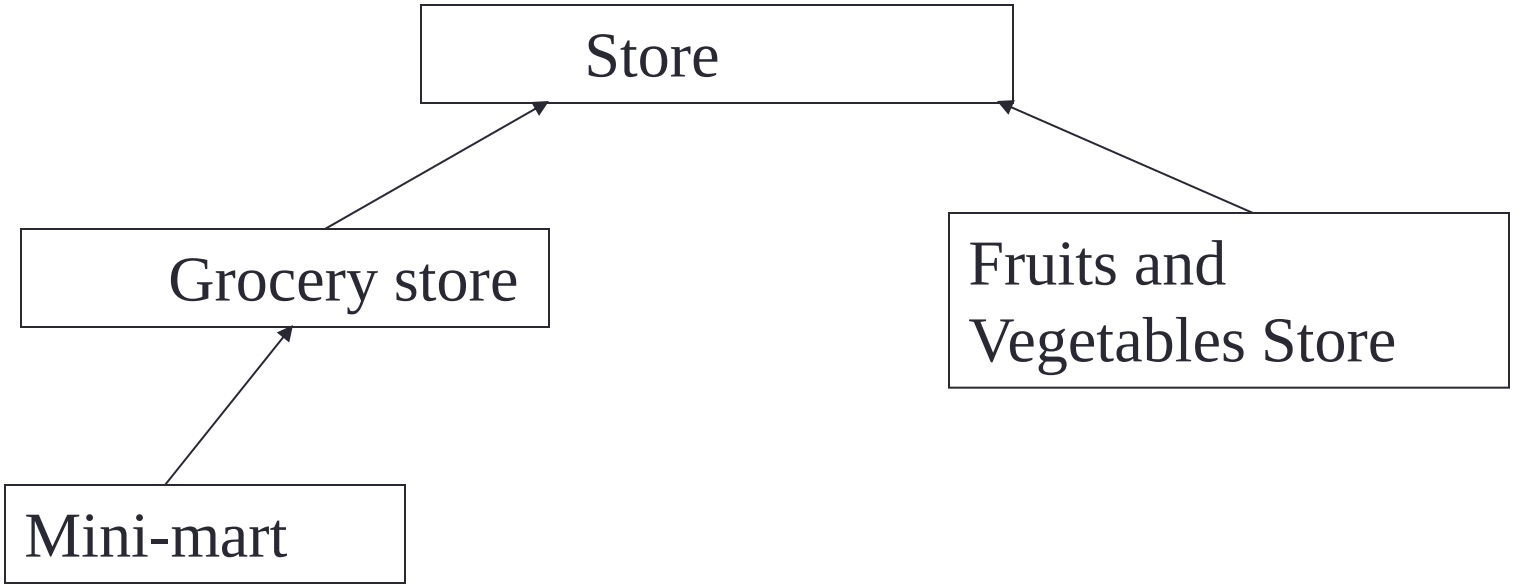
OOP RECAP

Inheritance



Is-A relationship

Inheritance

- An object of a subclass IS-A(n) object of the superclass
- Example:

Store s=new GroceryStore();

Note: In Java a class can only extend one superclass but more than one subclass can be derived from one superclass.

Has-A

- A GroceryStore Has-A shelf
- Has –A relationship means a class has a data field.
- You cannot say that a GroceryStore IS-A cashier(staff)

Subclass

- Inherits all *public methods of its superclass*
- *Superclass' private methods are NOT callable from its sublcasses.*
- Superclass has a method getName()

```
public class GroceryStore extends Store{

public void application() {
String applicName= getName(); // calls the superclass
                                  method getName() directly
```

Subclass client(objects)

GroceryStore gs= new GroceryStore();
String applicName=gs.getName();

A subclass object can call a super's method directly

public class GroceryStore extends Store {
public String toString() {
return super.getName() + staff()+…;

Constructors

Sublcass must provide its own or rely on the superclass “no argument” constructor

```
public class Store() {
private String name;
private int numEmployees;
public  Store(String nam, int empl){
Name=nam;
numEmployees=empl;}
```

```
public class GroceryStore extends Store {
private string location;
public GroceryStore(String nam, int empl, String loc) {
   super(nam,empl);
   location=loc;
}
```

If the superclass constructor is used it MUST BE the first line in the constructor.

If it is not used by default the no-argument constructor of the superclass is automatically called. No argument constructor must always be defined in the superclass to avoid errors.

Fields

- Instance variables(fields) should always be declared **private** in AP
- They are **not** accessible by the subclasses **directly**
- Modifier methods must be used (set/get)
- *Superclass public constants (public static final) are accessible everywhere*.
- A subclass can and does have its own static or instance variables

Abstract Classes

- Some of the methods in a class can be declared abstract and left with only signatures defined
- *A class with one or more abstract methods must be declared abstract*

```
public abstract class Organize
{
   ...
   public abstract void stockShelves();
}
```

Abstract classes (cont'd)

- Abstract classes serve as common superclasses for more specific classes
- Abstract classes are closer to the root of the hierarchy; they describe more abstract objects
- Java does not allow us to instantiate (that is, create objects of) abstract classes.
- An abstract class can have constructors — they can be called from constructors of subclasses

Class Object

- In Java every class by default extends a library class Object (from java.lang)
- Object is a concrete class (non abstract)

Interfaces

- An interface in Java is similar to an abstract class
- It Does Not Have any fields or constructors
- All its methods are abstract

```
public interface Learn
{
   String getName ();
   String getMarks (int m);
   int[] getSubjects();
}
```

Interfaces (cont'd)

- We must "officially" state that a class implements an interface.
- When a class implements an interface ***it must supply all the methods of that interface.***

```
public class School implements  Learn
{
   String getName (){ return "Student"};
   String getMarks (int m) { ..};
   int[] getSubjects() {..};
}
```

Interface (cont'd)

- A class can implement several interfaces.
- Like an abstract class, an interface supplies a *secondary data type* to objects of a class that implements that interface.

Classes

- A superclass provides a secondary data type to objects of its subclasses.
- An abstract class cannot be instantiated.

Interfaces

- An interface provides a secondary data type to objects of classes that implement that interface.
- An interface cannot be instantiated.

Classes

- A concrete subclass of an abstract class must define all the inherited abstract methods.
- A class can extend another class.
- A subclass can add methods and override some of its superclass's methods.

Interfaces

- A concrete class that implements an interface must define all the methods specified by the interface.
- An interface can extend another interface (called its *superinterface*) by adding declarations of abstract methods.

Classes

- A class can extend only one class.
- A class can have fields.
- A class defines its own constructors (or gets a default constructor).

Interfaces

- A class can implement any number of interfaces.
- An interface cannot have fields (rarely some public static final constants).
- An interface has no constructors.

Classes

- A concrete class has all its methods defined.
- An abstract class usually has one or more abstract methods.
- Every class is a part of a hierarchy of classes with Object at the top.

Interfaces

- All methods declared in an interface are abstract.
- An interface may belong to a small hierarchy of interfaces, but this is not very common.