Interface Inheritance

An introduction to the Java Programming Language

Produced by:

Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhan Drohan (sdrohan@wit.ie)

Waterford Institute *of* Technology

INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

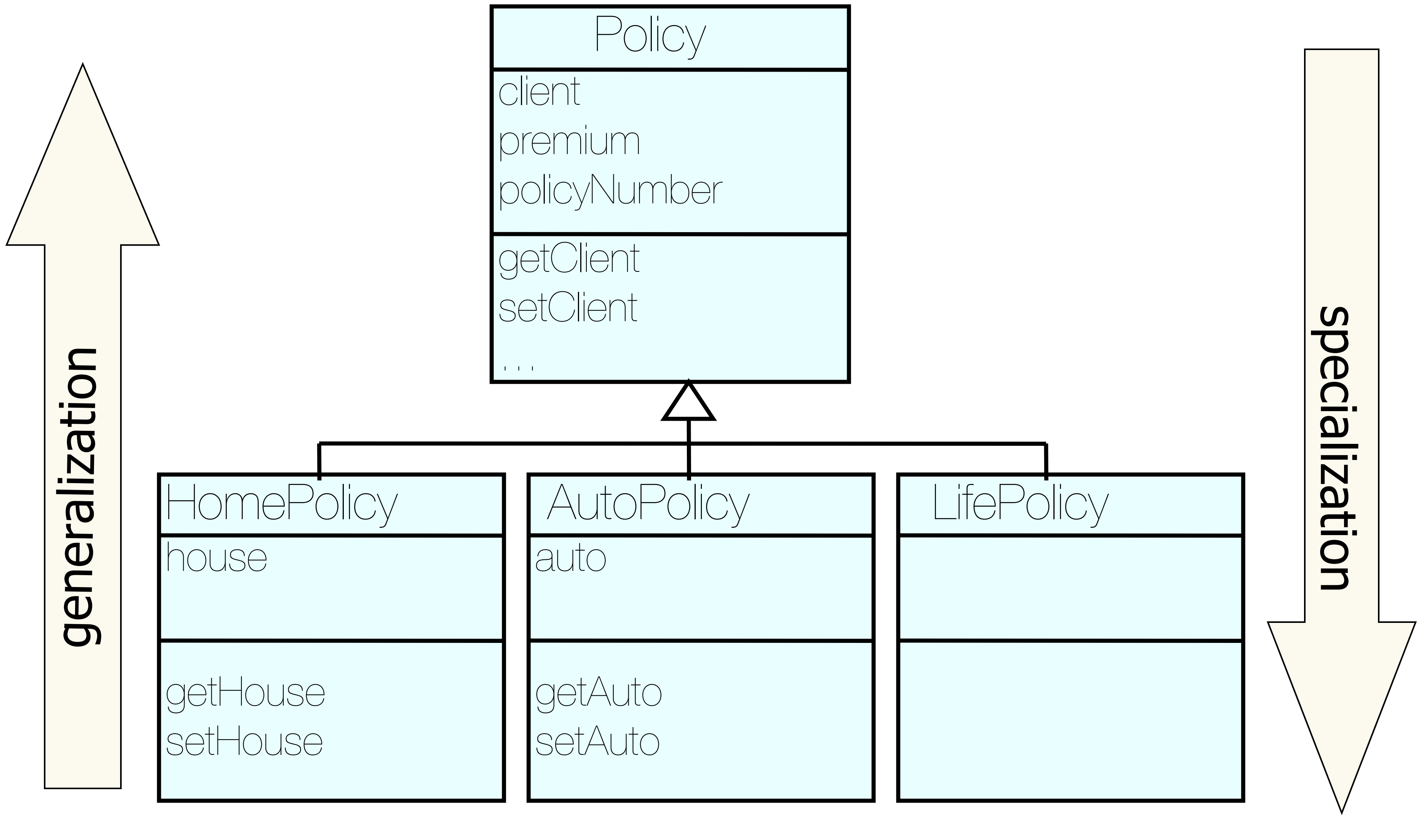
Agenda

- What is inheritance?
- Implementation Inheritance
    - Method lookup in Java
    - Use of this and super
    - Constructors and inheritance
    - Abstract classes and methods
- Interface Inheritance
    - Definition
    - Implementation
    - Type casting
    - Naming Conventions

Implementation vs Inheritance

Implementation Inheritance	Interface Inheritance
⌖ Promotes reuse. ⌖ Commonalities are stored in a parent class (superclass). ⌖ Commonalities are shared between children classes (subclasses).	⌖ Mechanism for introducing ***Types*** into java design. ⌖ Classes can support more than one interface, i.e. be of more than one ***type.***

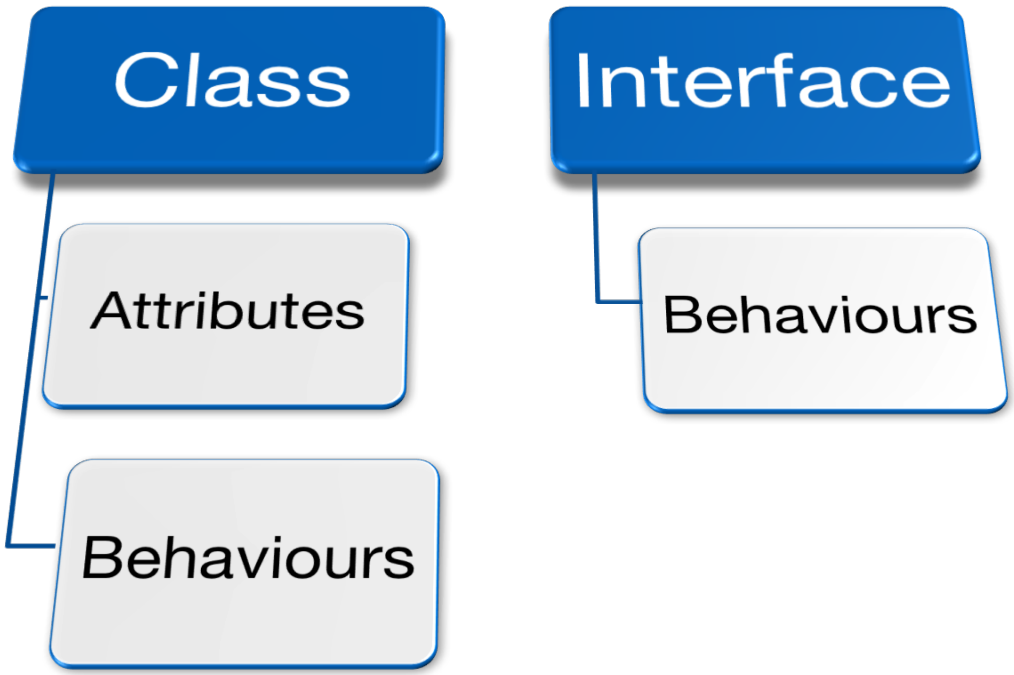
Implementation Inheritance



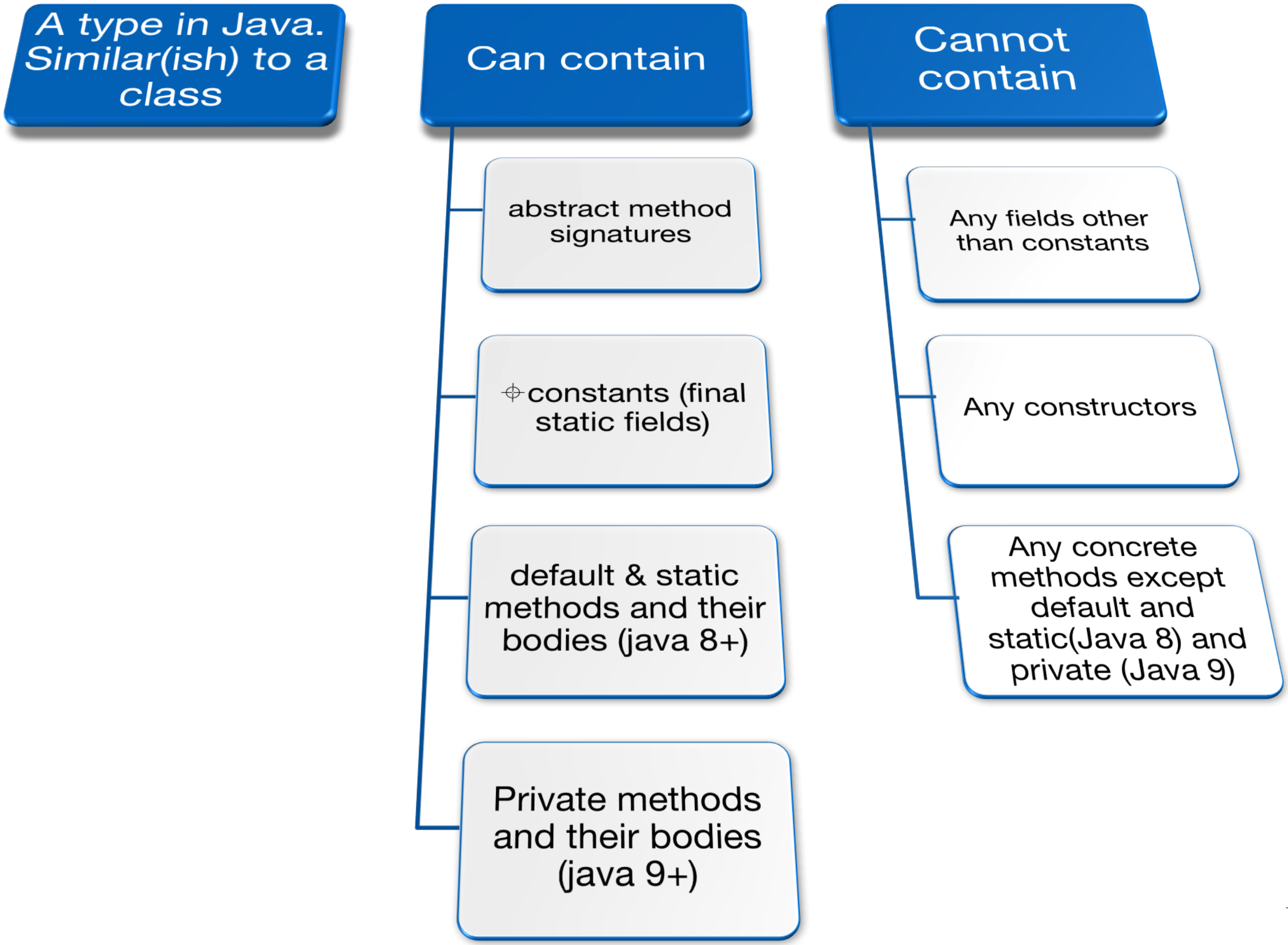
Overview: Road Map

- Interface Inheritance
  - Definition
  - Implementation
  - Type casting
  - Naming Conventions

What is an interface?



What is an interface?



A type in Java. Similar(ish) to a class
Can contain
abstract method signatures
constants (final static fields)
default & static methods and their bodies (java 8+)
Private methods and their bodies (java 9+)
Cannot contain
Any fields other than constants
Any constructors
Any concrete methods except default and static(Java 8) and private (Java 9)

Defining Interfaces – abstract methods

IAddressBook.java

```
public interface IAddressBook
{
  void clear();

  IContact getContact(String lastName);

  void addContact(IContact contact);

  int numberOfContacts();

  void removeContact(String lastName);

  String listContacts();
}
```

Methods are implicitly public and abstract

Defining Interfaces – abstract methods

IAddressBook.java

```
public interface IAddressBook
{
  void clear();

  IContact getContact(String lastName);

  void addContact(IContact contact);

  int numberOfContacts();

  void removeContact(String lastName);

  String listContacts();
}
```

Methods are implicitly public and abstract

NOTE: We will look at Java 8 and Java 9 Interface evolution in future lectures.

Overview: Road Map

- Interface Inheritance
  - Definition
  - Implementation
  - Type casting
  - Naming Conventions

Implementing Interfaces

```
public class AddressBook implements IAddressBook
{
  private Contact[]  contacts;
  private int nmrContacts;

  public AddressBook()
  {
    contacts = new Contact[IAddressBook.getCapacity()];
     nmrContacts = 0;
  }

 private int locateIndex(String lastName)
  {
    //…
  }
  public void clear(){
    //…
  }
…
}
```

```
public interface IAddressBook
{
  void clear();

  IContact getContact(String lastName);

  void addContact(IContact contact);

  int numberOfContacts();

  void removeContact(String lastName);

  String listContacts();
}
```

Implementing Interfaces

```
public class AddressBook implements IAddressBook
{
  private Contact[]  contacts;
  private int nmrContacts;

  public AddressBook()
  {
    contacts = new Contact[IAddressBook.getCapacity()];
     nmrContacts = 0;
  }

 private int locateIndex(String lastName)
  {
    //…
  }
  public void clear(){
    //…
  }
…
}
```

- Implementing classes <u>are subtypes</u> of the interface type.
- They <u>must</u> define all abstract methods for the Interface(s) they implement; otherwise the class must be declared abstract.

Implementing an Interface

⌖You can think of the class as **signing a contract**, agreeing to perform the specific behaviours of the interface.

A class can implement more than one interface at a time i.e. have more than one type.

A class can extend only one class, but implement many interfaces.

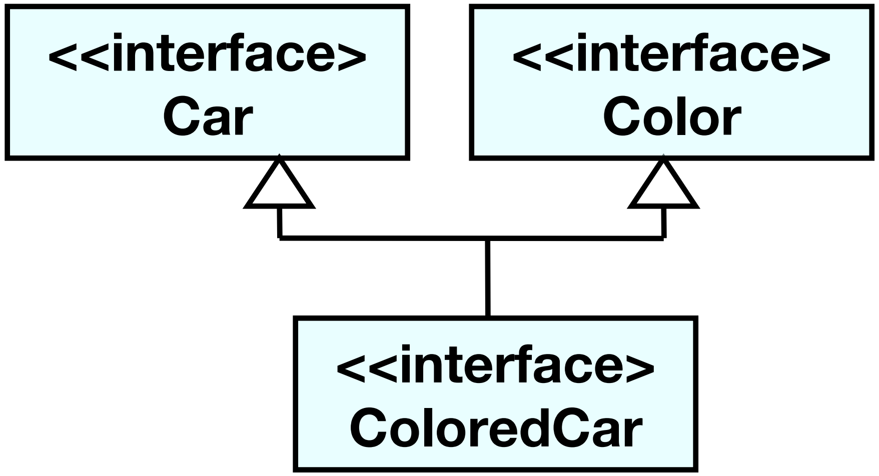
An interface ***can extend*** any number of interfaces (called subtyping). Multiple inheritance ***is*** allowed with interfaces.

An interface cannot implement another interface.

Extending Interfaces

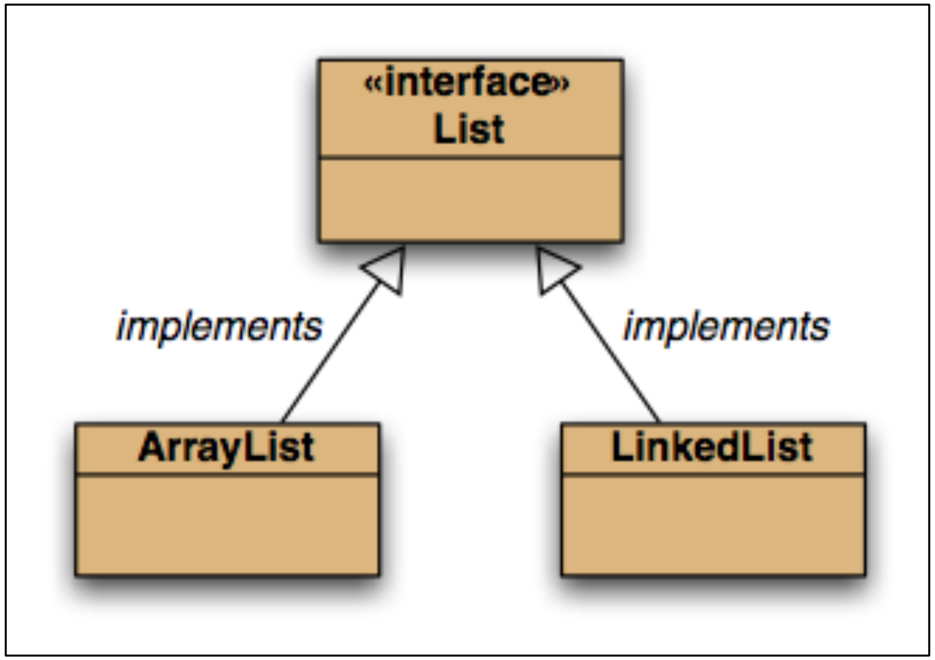
```
public interface Car
{
  public double getSpeed();
}
```

```
public interface Color
{
  public String getBaseColor();
}
```



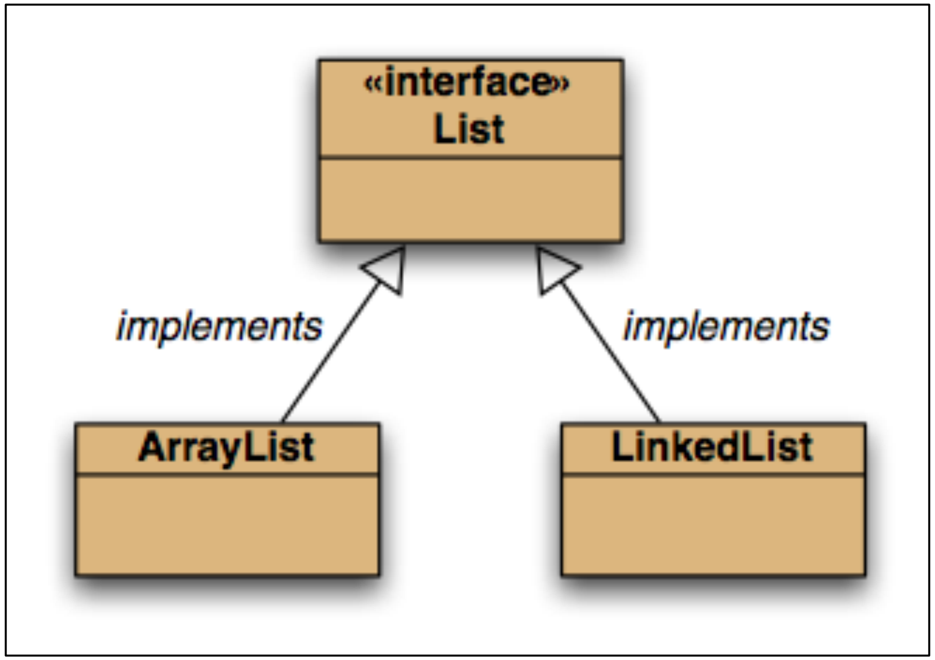
```
public interface ColoredCar extends Car, Color
{
   public String goFaster();
}
```

Interfaces in Collections Framework



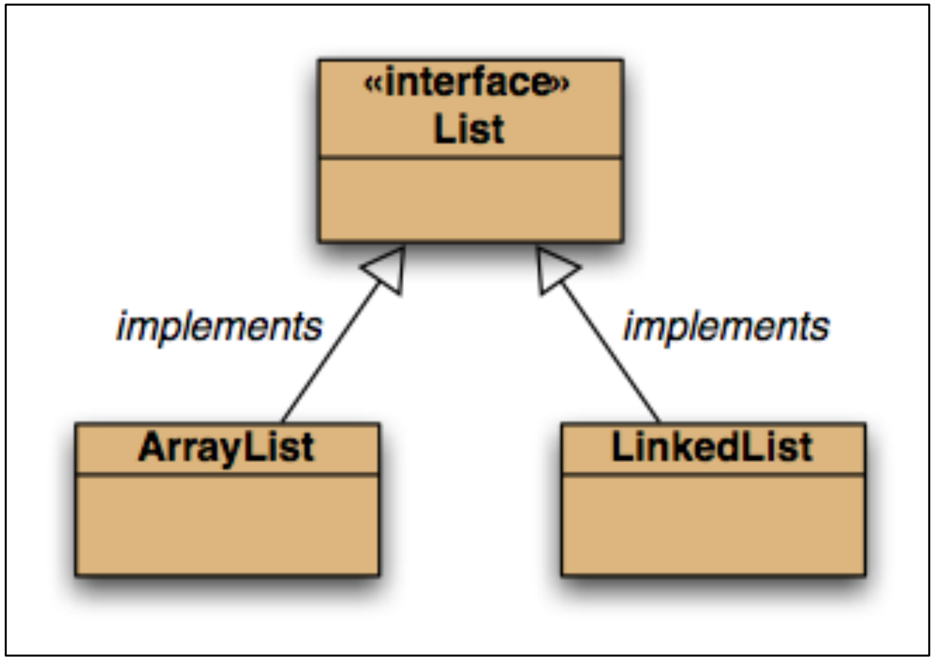
Interfaces in Collections Framework

If you define a reference/object variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface.



Interfaces in Collections Framework

If you define a reference/object variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface.



Applying this rule to a List:

```
List<Product> products = new ArrayList<Product>();
```

Overview: Road Map

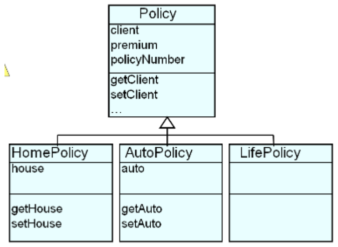
- Interface Inheritance
    - Definition
    - Implementation
    - Type casting
    - Naming Conventions

Reference vs Interface type

- Reference type
  - Any instance of that class or any of the subclasses can be assigned to the variable.

```
Policy policy;
policy = new Policy();
```

```
Policy policy;
policy = new HomePolicy();
```

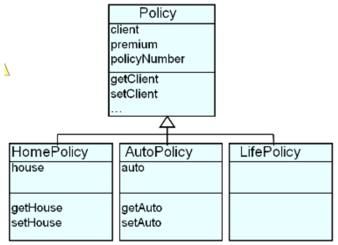


Reference vs Interface type

- Reference type
  - Any instance of that class or any of the subclasses can be assigned to the variable.

```
Policy policy;
policy = new Policy();
```

```
Policy policy;
policy = new HomePolicy();
```



- Interface type
  - Any instance of any class that implements that interface can be assigned to the variable.

```
IAddressBook book;

book = new AddressBook();
book.clear();
book.addContact(contact);
//… etc…
```

book declared as an **IAddressBook** interface type

Variables and Messages

- If a variable is defined as a certain type, only messages defined for that type can be sent to the variable.

```
IAddressBook book;

book = new AddressBook();

book.clear();
book.addContact(contact);

int i = book.locateIndex("mike");

// Error!
//
// static type is IAddressBook →
// compile-time check finds that
// locateIndex() is defined in
// AddressBook - but not in
// IAddressBook.
```

Type Casting

- Type casting can be subverted (undermined) by type checking.
- To be used rarely and with care.
- Type cast can fail, and run time error will be generated if the book object really is not an AddressBook
  - (e.g. it could be an AddressBookMap which also implements IAddressBook)

```
IAddressBook book;

book = new AddressBook();

book.clear();
book.addContact(contact);

int i = ((AddressBook)book).locateIndex(“mike”);
```

Type cast from IAddressBook to AddressBook

Common Naming Conventions

- There are a few conventions when naming interfaces:
  - Suffix **able** is often used for interfaces
    - Cloneable, Serializable, and Transferable
  - Nouns are often used for implementing classes names, and I + noun for interfaces
    - Interfaces: IColor, ICar, and IColoredCar
    - Classes: Color, Car, and ColoredCar
  - Nouns are often used for interfaces names, and noun+Impl for implementing classes
    - Interfaces: Color, Car, and ColoredCar
    - Classes: ColorImpl, CarImpl, and ColoredCarImpl

Agenda

- What is inheritance?
- Implementation Inheritance
  - Method lookup in Java
  - Use of this and super
  - Constructors and inheritance
  - Abstract classes and methods
- Interface Inheritance
  - Definition
  - Implementation
  - Type casting
  - Naming Conventions