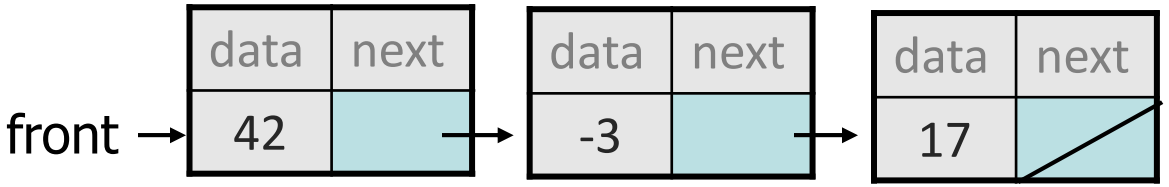
Abstract Classes

Example – List classes

Suppose we have implemented the following two list classes:

- `ArrayList`

- `LinkedList`



- We have a `List` interface to indicate that both implement a List ADT.
- Some of their methods are implemented the same way (redundancy).

Problem

- Every time we modify one "copy" of that code, we need to update all other versions – by hand!!!

So… Limits of Interfaces

Our List Classes share some Common code

- `add(`**value**`)`
- `contains`
- `isEmpty`

BUT Interfaces do not hold code

How can we capture this common behavior?

Inheritance to the rescue!

So… Limits of Interfaces

Our List Classes share some Common code

- `add(`**value**`)`
- `contains`
- `isEmpty`

BUT Interfaces do not hold code

Should we change our interface to a class? Why / why not?

Tradeoff!
Free inheritance slot vs. factor code

Abstract classes

Definition – A hybrid between an interface and a class.

- Like a superclass may contain both
  - method declarations (like an interface)
  - and/or method bodies (like a class)
- Like an interface, may not be instantiated
  (cannot use `new` to create any objects of their type)

What goes in an abstract class?

- implementation of common state and behavior that will be inherited by subclasses (parent class role)
- declare generic behaviors that subclasses have to implement (interface role)

Abstract class syntax

```
// declaring an abstract class
public abstract class name {
    ...

    // declaring an abstract method
    // (any subclass must implement it)
    public abstract type name(parameters);

}
```

Rules

- A class can be `abstract` even if it has no abstract methods
- You can create variables (but not objects) of the abstract type

Example – abstract list class

```
// Superclass with common code for a list of integers.
public abstract class AbstractIntList implements List {
    public void add(int value) {
        add(size(), value);
    }

    public boolean contains(int value) {
        return indexOf(value) >= 0;
    }

    public boolean isEmpty() {
        return size() == 0;
    }
}


public class ArrayIntList extends AbstractIntList { …

public class LinkedIntList extends AbstractIntList { …
```

Abstract vs. Normal Classes: how do they implement interfaces

- Normal classes that claim to implement an interface must implement all methods of that interface:

```
public class Empty implements List {}  // error
```

- Abstract classes may claim to implement an interface without implementing its methods; BUT subclasses must implement the methods.

```
public abstract class Empty implements List {} // ok

public class Child extends Empty {}            // error
```

Abstract classes vs. interfaces

Observation

- An abstract class can do everything an interface can do and more.

So…

- Why do both interfaces and abstract classes exist in Java?

Answer

- Java has single inheritance; i.e. one class may only extend only one superclass
- BUT it may implement many interfaces
- Having interfaces allows a class to be part of a hierarchy (polymorphism) without using up its inheritance relationship.

Abstract classes vs. Regular Classes

Observation

- Inheriting from an abstract class can do everything inheriting from a regular class does and more.

So…

- Why do we have both mechanisms?

Answer

- Abstract classes force subclasses to implement some methods
  - Unable to construct new object from an abstract class only
  - Unable to compile a subclass of our abstract class if it is not implementing all abstract methods
- As such they are a contract, like interfaces

Abstract classes vs. Regular Classes

Observation

- Inheriting from an abstract class can do everything inheriting from a regular class does and more.

So…

- Why do we have both mechanisms?

Answer

- Wait…
- Since we allow abstract classes to not have any abstract methods… then we could use these for everything!

- Not quite…
  - Remember that an abstract class may not be instantiated! ☺