Inheritance

Building a better stack

What if I want to build a better stack?

- Add functionality
- Improve efficiency

Building a better stack

What if I want to build a better stack?

- Option 1: implement stack again

```
class myBetterStack  implements Stack{
    // implement push, pop, and empty
    ...
}
```

- Useful when:

  Entirely new implementation (e.g., don't use an array, use fractional cascading on a buffered tree).

Building a better stack

What if I want to build a better stack?

- Add functionality
- Improve efficiency

Solutions:

- Implement from scratch
- Modify original class
- Copy-paste old code to new class

Building a better stack

Inheritance

- MySpecialStack is a subclass (child) of Stack
- Stack is the superclass(parent) of MySpecialStack

```
class MySpecialStack extends Stack{

      void newFunction(){
      ...
      }

}
```

Building a better stack

Inheritance

- Subclass has all the functionality of the parent!

```
class MySpecialStack extends Stack{

     void newFunction(){
     ...
     }
}
```

```
MySpecialStack stack = new MySpecialStack();

stack.push(7)
stack.newFunction();
```

Building a better stack

Inheritance

- Subclass has all the functionality of the parent!

```
class MySpecialStack extends Stack{

     void newFunction(){
     ...
     }
}
```

MySpecialStack is a Stack

```
MySpecialStack stack = new MySpecialStack();

stack.push(7)
stack.newFunction();
```

Inheritance

Subclass substitutivity

- If `TypeBase` is a parent of `TypeOne`

  `TypeOne` extends `TypeBase`

- If `TypeOne` implements `TypeBase`

  `TypeBase` is an interface

```
TypeOne first;

TypeBase B = first;
```

Building a better stack

Inheritance

- Subclass has all the functionality of the parent!

```
class MySpecialStack extends Stack{

     void newFunction(){
     ...
     }
}
```

```
MySpecialStack stack = new MySpecialStack();

stack.push(7)
stack.newFunction();
```

Building a better stack

Inheritance

- Subclass can override parent class

```
class MySpecialStack extends Stack{

      // @override
      void push(k){
            count++;
            specialPush(k);
      }
}
```

Building a better stack

Inheritance

- Subclass can override parent class

```
class MySpecialStack extends Stack{

      // @override
      void push(k){
            count++;
            super.push(k);
      }
}
```

Building a better stack

Inheritance

- Subclass can override parent class

```
class animal {

      void eat(){ ... }

      void sleep(){ ... }

      void talk(){
            System.out.println("Hello");
      }

}
```

Building a better stack

Inheritance

- Subclass can override parent class

```
class dog extends animal{

      // @override
      void talk(){
            System.out.println(“Woof”);
      }

}
```

Building a better stack

Inheritance

- Subclass can override parent class

```
class cat extends animal{

      // @override
      void talk(){
            System.out.println(“Meow”);
      }

}
```

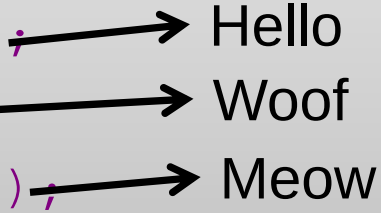
Building a better stack

Inheritance

- Subclass can override parent class

```
animal Alice = new animal();
animal Doug = new Dog();
animal Collin = new Cat();

Alice.talk();
Doug.talk();
Collin.talk();
```

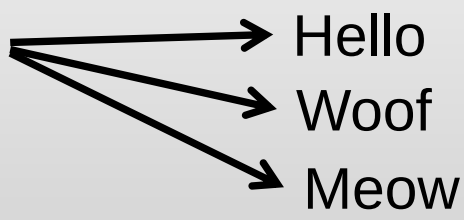


Building a better stack

Inheritance

- Subclass can override parent class

```
void pet(animal George){
     George.talk()
}
```



Building a better stack

Inheritance

- Subclass can override parent class

```
void pet(animal George){
     George.talk()
}



animal Doug = new Dog();
pet(Doug);
```

Building a better stack

Using a stack:

```
void fillStack(Stack store)
{
    for (int i=0; i<1000; i++)
    {
        store.push(i);
    }
}
```

```
{
    Stack A = new SlowStack()
    fillStack(A);
}
```

```
{
    Stack B = new FastStack()
    fillStack(B);
}
```

Building a better stack

Inheritance

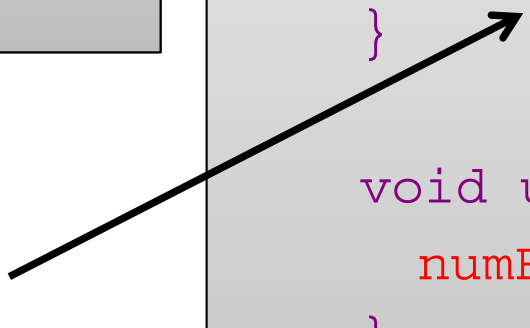
- Access: public, private, protected

```
class animal{
  private numEyes;
  protected numEars;
}
```

```
class dog extends animal{

   void updateEyes(){
     numEyes= 7;
   }

   void updateEars(){
     numEars= 10;
   }
}
```



Building a better stack

Inheritance

- Constructors are not inherited

```
class animal {
   public animal(int j){
     // Build your animal here
   }
}
```

```
class dog extends animal {
   public dog(int j){
      super(j);
   }
}
```

Building a better stack

Inheritance

- Default: child classes call empty parent constructor

```
class animal {
   public animal(int j){
     // Build your animal here
   }
}
```

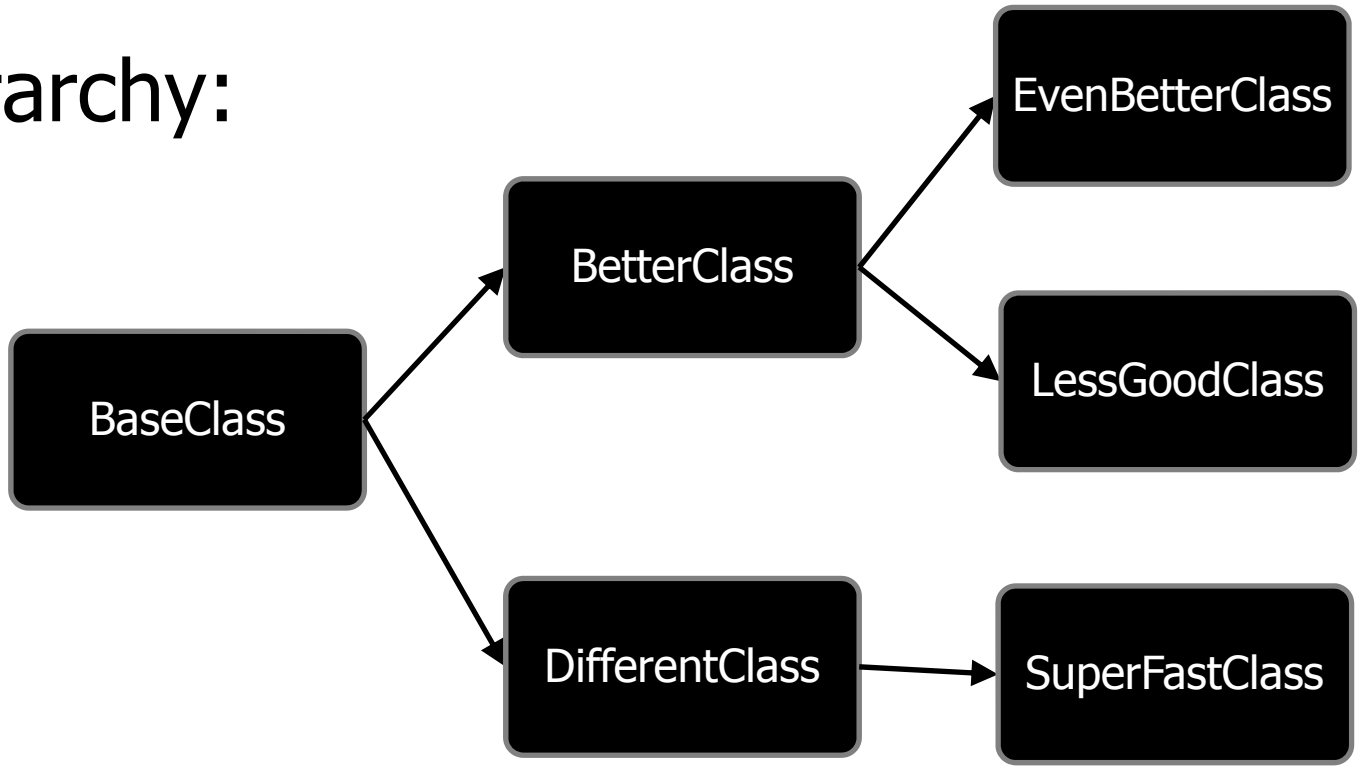
```
class dog extends animal {
   public dog(int j){
      super();
   }
}
```

Inheritance

Rules of inheritance:

- You can implement many interfaces.
- You can only extend one class.

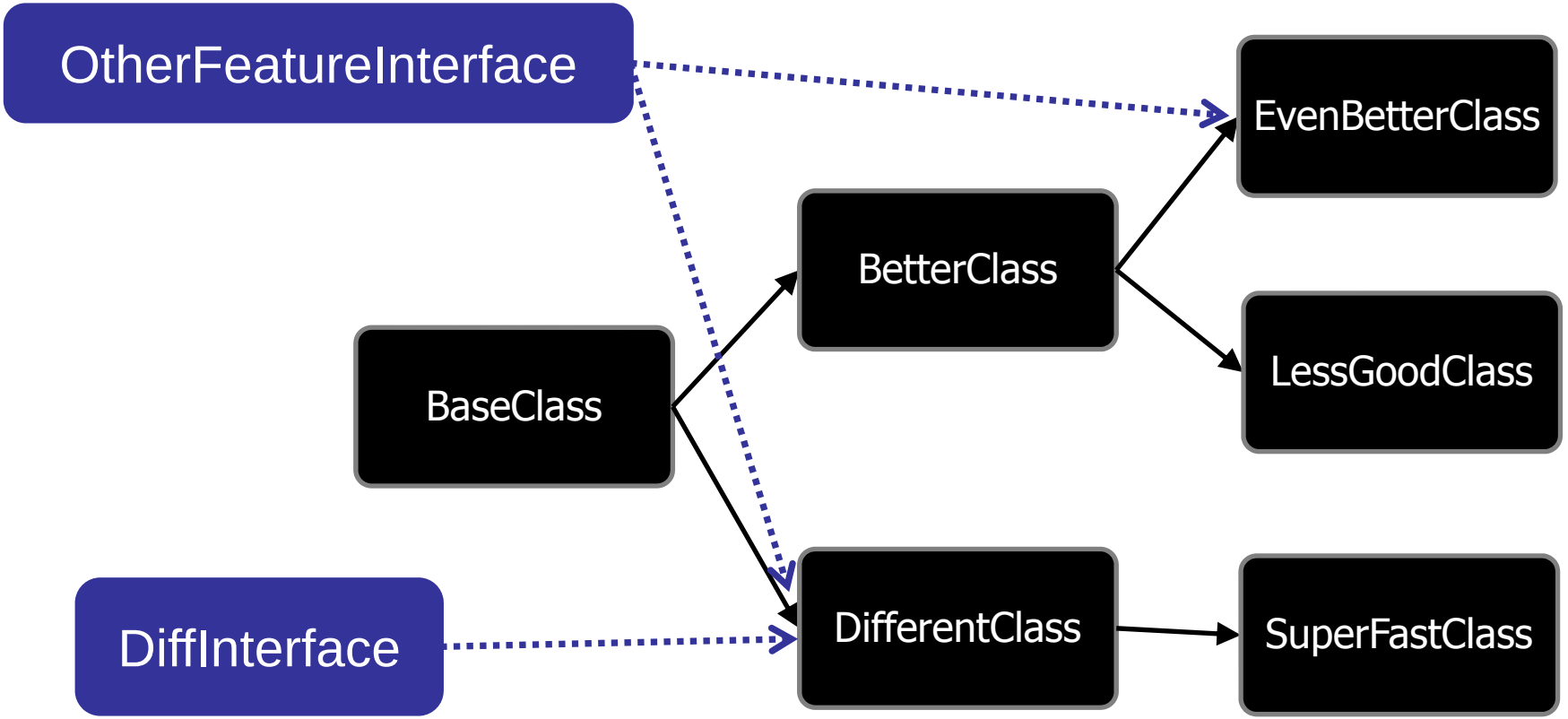
Class hierarchy:



Inheritance

Rules of inheritance:

- You can implement many interfaces.
- You can only extend one class.



Inheritance

VectorTextFile class:

- v1: slow
- v2: improved string management
- v3: improved sorting
- v4: no sorting

Problem:

- How to figure out what changed from v2 to v3?

Inheritance

VectorTextFile class:

- v1: slow
- v2: improved string management
- v3: improved sorting
- v4: no sorting

Good practice:

- Use inheritance!
- Each version contains only what is new.