

# CSE 12 – Basic Data Structures and Object-Oriented Design

## Lecture 3

Greg Miranda and Paul Cao  
Basic idea of ADT

# Announcements

- Quiz 3 due Friday @ 9am
- Survey 1 due Friday @ 11:59pm

# Topics

- Questions on Lecture 3?
- Interfaces

# Inheritance review: Which is legal?

```
public class Base
{
    protected int x;
}
```

```
public class Derived extends Base
{
    protected int y;
}
```

**A:**

```
Base b=new Base();
Derived d=b;
```

**B:**

```
Derived d=new Derived();
Base b=d;
```

**C:**

```
Base b=new Derived();
```

**D:**

```
Derived d=new Base();
```

**E:**

More than one of these

13) Given the following definitions:

```
public interface Printable
{
    public abstract String print( boolean duplex );
}
```

```
class Thing1 implements Printable
{
    private String str;

    public Thing1()
    {
        this.str = "Thing 1";
    }

    public String print( boolean duplex )
    {
        return this.str + " duplex = " + duplex;
    }

    public String print()
    {
        // print single sided by default
        return this.print( false );
    }
}
```

```
class Thing2 implements Printable
{
    private String str;

    public Thing2()
    {
        this.str = "Thing 2";
    }

    public String print( boolean duplex )
    {
        return this.str + " duplex = " + duplex;
    }

    public String print( String user )
    {
        System.out.print( user + ": " );

        // print double sided by default
        return this.print( true );
    }
}
```

And the following variable definitions:

```
Thing1 thing1 = new Thing1();
Thing2 thing2 = new Thing2();
Printable printable;
```

Hint: What does the compiler know about any reference variable at compile time (vs. run time)?

What gets printed with the following statements (each statement is executed in the order it appears). If there is a compile time error, write "Error" and assume that line is commented out when run.

```
System.out.println( thing1.print() );
System.out.println( thing1.print( "CS11SZZ" ) );
System.out.println( thing1.print( false ) );
System.out.println( thing2.print() );
System.out.println( thing2.print( "CS11SZZ" ) );
System.out.println( thing2.print( false ) );
printable = thing1;
System.out.println( printable.print( true ) );
System.out.println( printable.print() );
System.out.println( printable.print( "CS11SZZ" ) );
printable = new Thing2();
System.out.println( printable.print( true ) );
System.out.println( printable.print() );
System.out.println( printable.print( "CS11SZZ" ) );
```

*Thing1 duplex false*

*compiler error*

*Thing1 duplex false*  
*compiler error*

*CS11SZZ: Thing2 : duplex true*

*Thing2 : duplex true*

*Thing1 : duplex true*  
*compiler error*

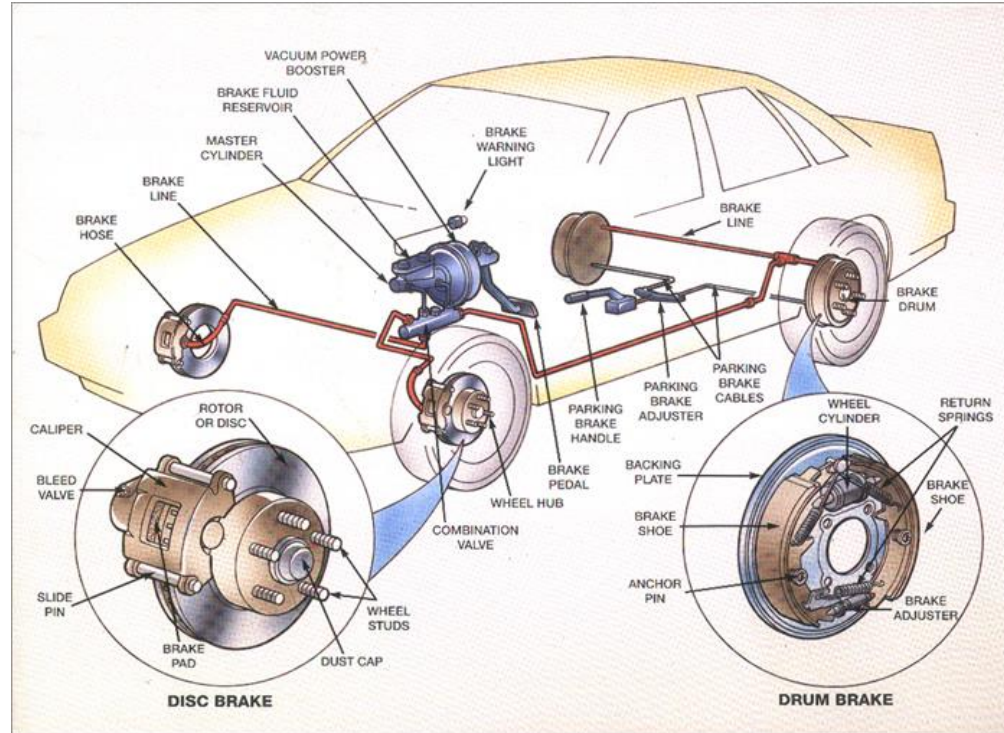
*compiler error*

*Thing2 : duplex true*  
*compiler error*

*compiler error*

Key idea in designing code:  
Data Abstraction

# Abstraction example: car brakes



# ADT Implementers and Users

## Implementers



“We can implement the ADT however we want!”

**ADT Interface:**  
sets the rules of interaction

## Users

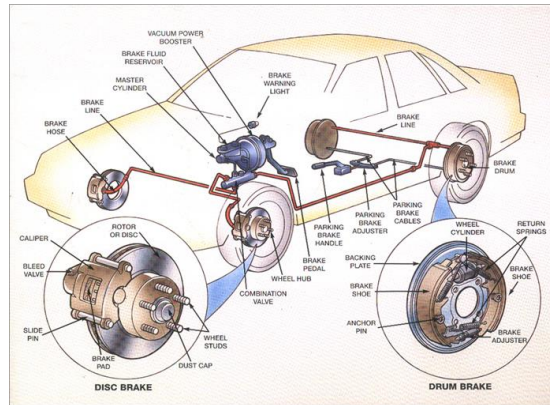


“We can use the ADT however we want!”



# ADTs vs specific implementations

- Sometimes the line between the abstract data type and the implementation is confusing. E.g. Car brakes



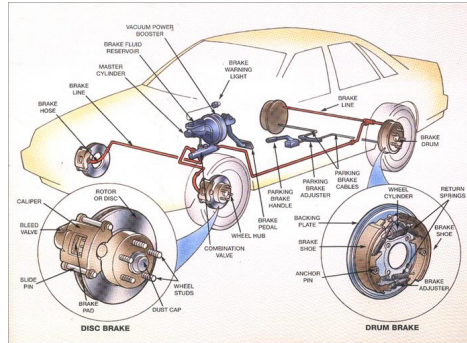
To drive my car, I rely on the abstract idea of brakes.

But for that idea to work, my car needs to have a specific implementation of brakes.

Someone needs to care about both sides!

# ADTs vs specific implementations

- It usually depends on which side you're looking from. e.g. car brakes



In this class (and as a computer scientist) you will be the driver AND the mechanic! (i.e. the designer and the user)

# ADTs vs APIs

ADT = "Abstract Data Type"

Defines the behaviors of a data type, but NOT its implementation

API = "Application Programming Interface"

Specifies public methods for interacting with a library or class

Does NOT reveal implementation details

Think of it as a language/code-specific ADT

# Exercise

- Please create an interface for Animals on Earth
  - Think what functionalities earth creatures share (dog, cat, dophin, human, etc)
  - Why don't we make it an abstract class? → more freedom.