Abstract Classes and Interfaces

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688P, Fall Semester 2020

October 12, 2020

Overview

Part 2

Quick Review

Quick Reiew: Objects and Classes

Working with Objects and Classes:

- Collections of objects share similar traits (e.g., data, structure, behavior).
- Collections of objects will form relationships with other collections of objects.

Definition of a Class

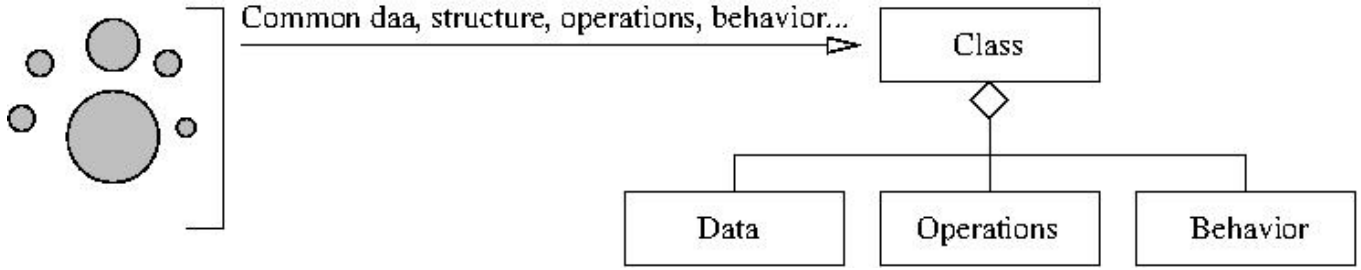
A class is a specification (or blueprint) of an object's structure and behavior.

Definition of an Object

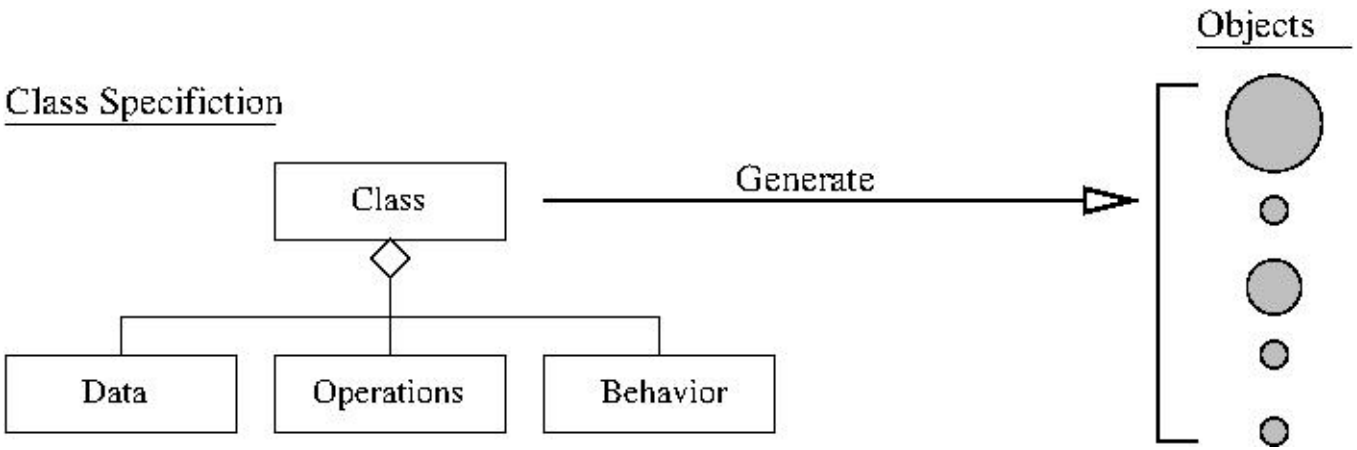
An object is an instance of a class.

Quick Review: Objects and Classes

From Collections of Objects to Classes:



Generation of Objects from Class Specifications:



Quick Review: Objects and Classes

Key Design Tasks

- Identify objects and their attributes and functions,
- Establish relationships among the objects,
- Establish the interfaces for each object,
- Implement and test the individual objects,
- Assemble and test the system.

Implicit Assumptions → Connection to Data Mining

- Manual synthesis of the object model is realistic for systems that have a modest number of elements and relationships.
- As the dimensionality of the problem increases some form of automation will be needed to discover elements and relationships.

Working with Interfaces

Programming to an Interface

Motivation

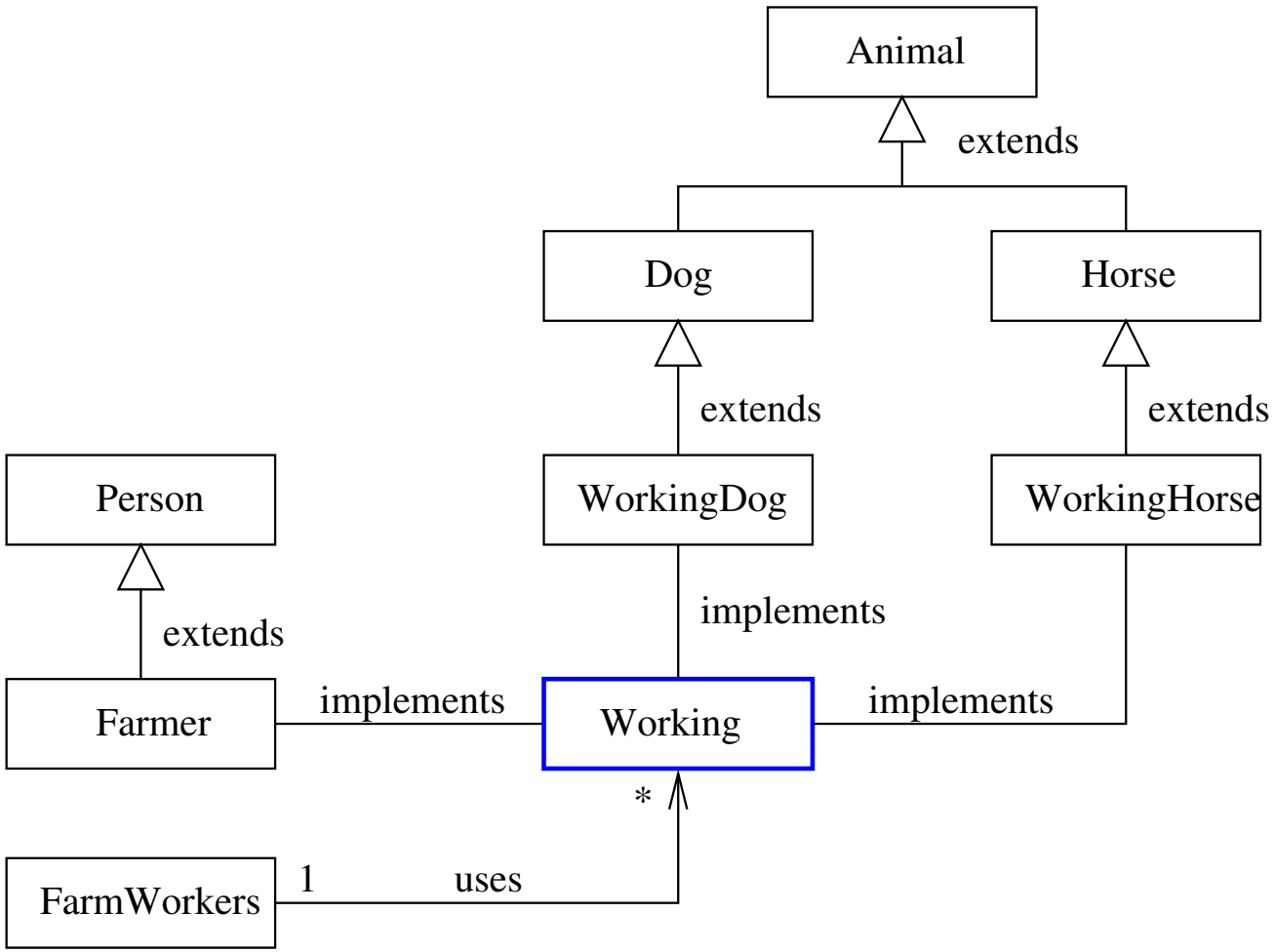
- Interfaces are the mechanism by which components describe what they do, but not how they do it.
- Interface abstractions are appropriate for collections of objects that provide common functionality, but are otherwise unrelated.

Implementation

- An interface defines a set of methods without providing an implementation for them.
- An interface does not have a constructor – therefore, it cannot be instantiated as a concrete object.
- Any concrete class the implements the interface must provide implementations for all of the methods listed in the interface.

Working with System Interfaces

Example 1. Software Interface for Farm Workers



Working with System Interfaces

Example 1. Software Interface for Farm Workers

Workers is simply an abstract class that defines an interface, i.e.,

```
public interface Working {
    public abstract void hours ();
}
```

In Java, the interface is implemented by using the keyword implements in the class declaration, e.g.,

```
public class Farmer implements Working { ....
```

This declaration sets up a contract that guarantees the Farmer class will provide a concrete implementation for the method hours().

Working with System Interfaces

Important Point. Instead of writing code that looks like:

```
Farmer       mac = new Farmer (...);
WorkingDog   max = new WorkingDog (...);
WorkingHorse silver = new WorkingHorse (...);
```

We can treat this group of objects as a set of Working entities, i.e.,

```
Working    mac = new Farmer (...);
Working    max = new WorkingDog (...);
Working silver = new WorkingHorse (...);
```

Methods and algorithms can be defined in terms of all Working entities, independent of the lower-level details of implementation.

Programming to an Interface

Motivation and Benefits

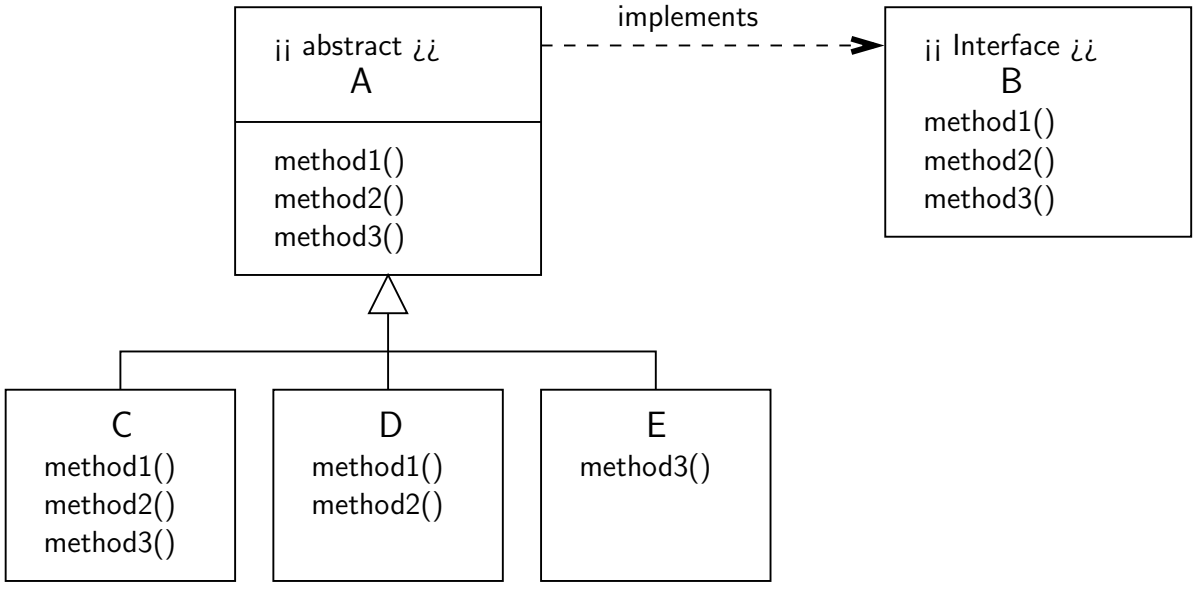
In Java, an interface represents what a class can do, but not how it will do it, which is the actual implementation.

Two key benefits:

- Information hiding. As long as the objects conform to the interface specification, then there is no need for the clients to know the exact type of the objects they use.
- Improved flexibity. System behavior can be changed by swapping the object used with another implementing the same interface.

Programming to an Interface

Combining Abstract Classes and Interfaces



Now we can write:

```
Creating objects of type C,D and E.   Executing methods ...
===================================== ====================
B c1 = new C (...);                   c1.method1();
B d1 = new D (...);                   d1.method2();
B e1 = new E (...);                   e1.method3();
===================================== ====================
```

Farm Worker Source Code

Working with System Interfaces

Source Code: Animal.java

```
1  public class Animal {
2      String name;
3
4      public Animal( String name ) { this.name = name; }
5      public String toString()     { return this.name; }
6  }
```

Source Code: Dog.java

```
1  public class Dog extends Animal {
2      public Dog( String name ) { this.name = name; }
3
4      public String toString(){
5         return "*** In Dog: " + this.name;
6      }
7  }
```

Source Code: Horse.java

```
1  public class Horse extends Animal {
2      public Horse( String name ) { this.name = name; }
3
4      public String toString() {
5         return "*** In Horse: " + this.name;
6      }
7  }
```

Working with System Interfaces

Source Code: WorkingDog.java

```
1  public class WorkingDog extends Dog implements Working {
2      public WorkingDog( String name ) {
3         this.name = name;
4      }
5
6      public void hours () {
7         System.out.println ( "*** Working dog hours -- working weekends!!" );
8      }
9  }
```

Source Code: WorkingHorse.java

```
1  public class WorkingHorse extends Horse implements Working {
2      public WorkingHorse( String name ) {
3         this.name = name;
4      }
5
6      public void hours () {
7         System.out.println ( "*** Working horse hours -- also working weekends!!" );
8      }
9  }
```

Source Code: Working.java (Interface)

```
1  public interface Working {
2     public abstract void hours ();
3  }
```

Working with System Interfaces

Source Code: Person.java

```
1   /*
2    * ================================================================
3    * Person.java. Create person objects and compute their age...
4    *
5    * Written By: Mark Austin                                December 2006
6    * ================================================================
7    */
8
9   import java.util.Calendar;
10  import java.util.Date;
11  import java.util.GregorianCalendar;
12
13  public class Person {
14     protected String   sName;
15     protected Date birthdate;
16
17     // ================================================================
18     // Set/get name of a person
19     // ================================================================
20
21     public void setName( String sName ) {
22        this.sName = sName;
23     }
24
25     public String getName() {
26        return sName;
27     }
```

Working with System Interfaces

Source Code: Person.java (continued)

```
28
29      // ============================================================
30      // Compute age of a person ...
31      // ============================================================
32
33      public int getAge() {
34          ... details removed ...
35      }
36
37      public void setBirthDate(Date aBirthDate) {
38          this.birthdate = aBirthDate;
39      }
40
41      public void setBirthDate(int iYear, int iMonth, int iDay ) {
42          Calendar cal = Calendar.getInstance();
43          cal.set( iYear, iMonth, iDay );
44          this.birthdate = cal.getTime();
45      }
46
47      public Date getBirthDate() {
48          return birthdate;
49      }
50
51      // ============================================================
52      // Create a String description of a persons cridentials
53      // ============================================================
```

Working with System Interfaces

Source Code: Person.java (continued)

```
54
55      public String toString() {
56         String s = "Name: " + getName() + "\n";
57               s += " Age: " + getAge() + "\n";
58         return s;
59      }
60   }
```

Source Code: Farmer.java

```
1   public class Farmer extends Person implements Working {
2       public Farmer() {
3          super();
4       }
5
6       public Farmer( String name ) {
7          super();
8          this.sName = name;
9       }
10
11      public String toString() {
12         return "*** In Farmer: " + this.sName;
13      }
14
15      public void hours () {
16         System.out.println ( "*** Working farmer -- working 7 days a week!!" );
17      }
18  }
```

Working with System Interfaces

Source Code: FarmerWorkers.java (Test Program) ....

```
1   public class FarmWorkers {
2      public static void main ( String args[] ) {
3
4          // Create objects for farmers ....
5
6          Working mac = new Farmer( "Old MacDonald" );
7          System.out.println( mac.toString() );
8          mac.hours();
9
10         // Create objects for working farm animals ..
11
12         Working max = new WorkingDog( "Max" );
13         System.out.println( max.toString() );
14         max.hours();
15
16         Working silver = new WorkingHorse( "Silver" );
17         System.out.println( silver.toString() );
18         silver.hours();
19     }
20  }
```

Working with System Interfaces

Test Program Output:

```
*** In Farmer: Old MacDonald
*** Working farmer -- working 7 days a week!!
*** In Dog: Max
*** Working dog hours -- working weekends!!
*** In Horse: Silver
*** Working horse hours -- also working weekends!!
```

You might wonder:

Can I use this approach to call methods that are within a participating class (e.g., WorkingHorse), but not defined in the interface?

- No! You can only call methods defined in the interface.