CS120: Interfaces

W. Michael Petullo

University of Wisconsin–La Crosse

As of October 6, 2021

Interfaces: Two Similar Classes

```
class Host {
  private String name;

  public Host(String name) {
    this.name = name;
  }

  public void greet() {
    ...println("Welcome!");
  }
}
```

```
class Guest {
  private String name;

  public Guest(String name) {
    this.name = name;
  }

  public void greet() {
    ...println("I am " + name);
  }
}
```

- A class definition creates a lifeless category.
- A `main` method breaths life into a class to generate one or more objects.
- The `main` method and the methods that `main` calls bring action to the program.

Interfaces: Two Similar Classes

```
class Host {
  private String name;

  public Host(String name) {
    this.name = name;
  }

  public void greet() {
    ...println("Welcome!");
  }
}
```

```
class Guest {
  private String name;

  public Guest(String name) {
    this.name = name;
  }

  public void greet() {
    ...println("I am " + name);
  }
}
```

```
class Party {
  public static void main(String[] args) {
    Host  h = new Host("Eddie");
    Guest g = new Guest("Angus");
    h.greet();
    g.greet();
  }
}
```

Interfaces: Unifying Two or More Similar Classes

```
interface Attendee {
  public void greet();
}
```

```
class Host2 implements Attendee {
  private String name;

  public Host2(String name) {
    this.name = name;
  }

  public void greet() {
    ...println("Welcome!");
  }
}
```

```
class Guest2 implements Attendee {
  private String name;

  public Guest2(String name) {
    this.name = name;
  }

  public void greet() {
    ...println("I am " + name);
  }
}
```

Interfaces: Polymorphism

```
public class Party {
  public static void main(String[] args) {
    Host  h = new Host("Eddie");
    Guest g = new Guest("Angus");
    h.greet();
    g.greet();
  }
}
```

vs.

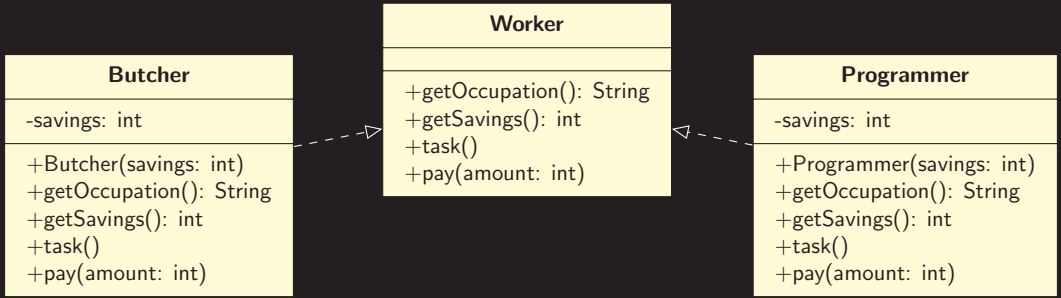
```
public class Party2 {
  public static void main(String[] args) {
    Attendee[] a = new Attendee[3];
    a[0] = new Host2("Eddie");
    a[1] = new Guest2("Angus");
    a[2] = new Guest2("Slash");

    for (int i = 0; i < a.length; i++) {
      a[i].greet();
    }
  }
}
```

Top example assigns `Host` and `Guest` objects to variables of type `Host` and `Guest`, respectively.

Bottom example assigns `Host` and `Guest` objects to (array) variable of type `Attendee`. Use of `interface` allows us to collect similar classes or pass similar classes to methods.

Interfaces: Busytown UML



The classes `Butcher` and `Programmer` implement the interface `Worker`. This means that you can assign an object of type `Butcher` or `Programmer` to a variable of type `Worker`. This will invoke `Programmers`'s `task` method despite `w` being a `Worker`:

```
Worker w = new Programmer();
w.task();
```

Interfaces: Defining and Implementing an Interface

```
interface Worker {
        ...
}

class Programmer implements Worker
    {
        ...
}

class Busytown {
        ...
}
```

Your `Busytown.java` should contain at least five classes (three illustrated here), and you must define each class independently. The classes should not nest.

Inheritance: Assignments

Graded Homework Aquinas: busytown and busytown2 in Java

Ungraded Labs Aquinas: vehicle and vehicle2 in Java

Reading Read Chapter 14

`https://www.flyn.org/courses/cs120-2021-fall/schedule`