Lesson 10

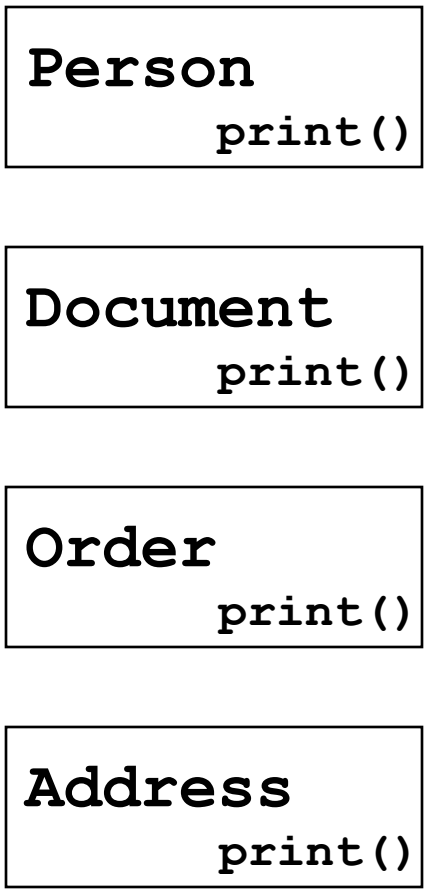
Work with Interfaces and Abstract Classes

Lesson Objectives

Upon completion of this lesson you should be able to

- Create an interface
- Implement an interface with a class
- Create an abstract class
- Extend an abstract class with a concrete class

What is an Interface



If all these classes have a **`print()`** method, and you want to use polymorphism, what is the common superclass?

Use an **Interface** to capture shared behavior

Use a **superclass** to capture shared data

Define an Interface

An interface

- defines methods that one or more classes will implement
- is a way to define shared behavior
- does not contain any implementation
- may have final constants with values
- is a **contract** between the designer and user

```
public interface Printable {
   public void print();
```

```
public class Person implements Printable {
  public void print(){….}
```

Design with Interfaces

- Application design can be more stable and flexible if designed with interfaces
- Determine desired behavior and put these methods in an interface
- Create object references that point to objects that support that interface
- Later implement the interface with an actual class

Interface: TimeTellable

Classes:
All implement TimeTellable, but all do it differently

Example Designing with Interfaces

```
public interface SpellChecker{
  public void getWord(String s);
  public String suggestWord();
}

public class WordProcessor {
  SpellChecker sp;
  sp = obj.getSpellChecker();
  sp.getWord("someString");
}
```

- Define desired spellchecker behavior at design time
- Can create an object reference that points to an object that implements an interface
- Passes at compile time
- At runtime will need an actual object that implements SpellChecker interface

Polymorphism with Interfaces

```
public interface Printable {
    public void print();
```

```
Person implements Printable
        print()
```

```
Document implements Printable
        print()
```

```
Order implements Printable
        print()
```

```
Address implements Printable
        print()
```

```
aMethod(Printable p){
  p.print();
}
```

Any of these objects can be passed to the method to be printed

Design with Interfaces

```
interface SelfDestructable{
 public void blowUp();
}
```

```
interface Flyable extends

       SelfDestructable{
 public void takeOff();
 public void crash();
 public void land();
}
```

```
interface Boatable extends
       SelfDestructable{
 public void skipSurface();
 public void sink();
}
```

```
public class JamesBondCar implements
                Flyable, Boatable{
 public void blowUp(){}
 public void takeOff(){}
 public void crash(){}
 public void land(){}
 public void skipSurface(){}
 public void sink(){}
```

```
public class Horsefly implements
                         Flyable {
 public void takeOff(){}
 public void crash(){}
 public void land(){}
 public void blowUp(){};
```

```
public class Rock implements
                         Boatable {
 public void blowUp(){}
 public void skipSurface(){}
 public void sink(){}
```

What about Inheriting Data and Methods?

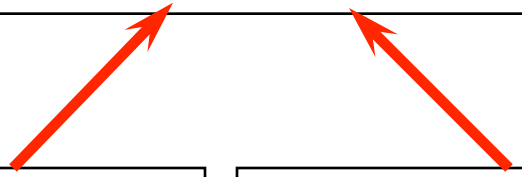
Class Vehicle is never actually instantiated:

- We would only instantiate a car, motorcycle or some other concrete subclass
- Vehicle can provide several benefits:
  1. Place to factor out common behavior and methods
  2. Place holder for empty methods that will have the implementation written in subclasses
  3. Enable polymorphism

Defining Abstract Classes in Java

- Use the `abstract` keyword to declare a class as abstract

```
public abstract class AbstractVehicle {
  private String model;
  public String getModel()…
}
```



```
public class Motorcycle
       extends
AbstractVehicle{
 private int nbrBags;
 public int getNbrBags(){…}
```

```
public class Car
       extends
AbstractVehicle{
private float trunkSize;
public float getTrunkSize(){…}
```

Define Abstract Methods

- An abstract method must be overridden by a concrete subclass
- An abstract method is a place-holder method and contains no code

```
public abstract class AbstractVehicle {
       // must be overridden in sublcasses
public abstract float calcMPG();
}
```

```
public class Motorcycle extends AbstractVehicle{
       // override from superclass
public float calcMPG(){…}
}
```

Abstract Methods

- An abstract method is one that cannot meaningfully be implemented by a class
  - a generic operation - a place holder
  - part of an abstract class
- Must be implemented by a concrete subclass
  - Each concrete subclass can implement the method differently
- An abstract method must be defined in an abstract class

Polymorphism with Abstract Classes

- Polymorphic collections can be defined using abstract classes

```
public abstract class AbstractVehicle {
  public abstract float calcMPG();
  …
```

```
AbstractVehicle[] vehicles = {
      new Motorcycle(…),
      new Car(…), …
};
…
for (int i=0; i < vehicles.length; i++)
  if (vehicles[i].calcMPG()) …
```

Using `instanceof` with Interfaces

- The `instanceof` operator can be used to check if an object implements an interface
- Downcasting can be used if necessary, to call methods defined in the interface

```
public void aMethod(Object obj) {
   …
   if (obj instanceof Printable)
      ((Printable)obj).print();
}
```

Summary

- An interface is a compiler enforced contract between the designer and user
- An interface holds behavior that is implemented by dissimilar classes
- Interfaces support polymorphism
- Abstract classes must be extended by a concrete class to be instantiated