Java OOP Review

Professor Frank Kreimendahl

School of Computing and Data Science

Wentworth Institute of Technology

Sep 7, 2022

ADTs

Abstract Data Types (ADTs)

ADT features:

- Uses **encapsulation** to limit access to data and methods
- Describes *what* a Java class does without describing *how*
- Supports the three fundamental operations of data: input, manipulation, output
- Allows for conceptually similar data and methods to be grouped
- Separates theory and implementation
- Gives structure to data – hence the course name!

ADT requirements:

- Never loses or forgets data (unless specified by an operation)
- Never duplicates data (unless specified by an operation)
- Functions consistently

Interfaces

Interface

An interface describes an ADT to the "user"

- the method names
- the parameter types to pass to methods
- the types of values returned from the methods
- the expected behavior of the methods

An interface is considered a contract – any ADT with an interface is guaranteed to have the interface's methods.

Interfaces With Classes

- A class that implements an interface provides code for the ADT – a description *how* the data is handled
- As long as the class satisfies the ADT contract, it may use any internal implementation
- The class may optionally include other public fields or methods as long as the interface is complete
  - data fields not necessary for the interface
  - methods not required for the interface
  - constructors (which cannot be specified in the interface)

Interface Example: ATM

An automated teller machine machine (ATM machine) allows a user to perform specified banking operations. Every machine provides operations to:

- verify the user's identity with a PIN
- allow the user to choose a particular account of theirs
- specify an amount of money to withdraw
- display the result of an operation
- display the balance of an account

Any Java class that implements an ATM machine must provide a method for each of these operations.
You can use an ATM without knowing the internal details.

ATM Interface

Declaration of a Java interface:

```
public interface ATM {
  /* method headers will go in here */
}
```

ATM Interface: Verify

Verify the user's identity with a PIN:

```
/** Verifies a user's PIN.
    @param pin The user's PIN
*/
boolean verifyPIN(String pin);
```

ATM Interface: Choose Account

Allow the user to choose a particular account of theirs

```
/** Allows the user to select an account.
    @return a String representing
            the account selected
*/
String selectAccount();
```

ATM Interface: Withdraw

Specify an amount of money to withdraw:

```
/** Withdraws a specified amount of money
    @param account The account from which
           the money comes
    @param amount The amount of money withdrawn
    @return whether or not the operation
            is successful
*/
boolean withdraw(String account, double amount);
```

ATM Interface: Display Results

Display the result of an operation:

```
/** Displays the result of an operation
    @param account The account from which money
           was withdrawn
    @param amount The amount of money withdrawn
    @param success Whether or not the withdrawal
           took place
*/
void display(String account, double amount,
             boolean success);
```

ATM Interface: Display Balance

Display the balance of an account:

```
/** Displays an account balance
    @param account The account selected
*/
void showBalance(String account);
```

ATM Interface: Full Interface

The full ATM interface:

(comments removed for brevity)

```
public interface ATM {

  boolean verifyPIN(String pin);
  String selectAccount();
  boolean withdraw(String account, double amount);
  void display(String account, double amount,
               boolean success);
  void showBalance(String account);

}
```

Interface Rules

- Only method headings are shown
- Interface methods are considered **abstract methods**
- abstract methods must be defined in every class that implements the interface
- constants may be declared in an interface, using a normal Java variable declaration/assignment

`implements` Keyword

If you want a class to implement an interface, use the `implements` keyword:

```
public class ATMbankAmerica implements ATM {
```

A class may implement multiple interfaces – interface names are comma-separated

Inheritance

- A Human *is* a Mammal – with more specific details
- Human has all the data fields and methods defined by Mammal
- Mammal is the **superclass** of Human
- Human is a **subclass** of Mammal
- Human can define more fields and methods (or redefine Mammal methods)
- A VesperBat is a Mammal – with different specifics than a Human
- Both Human and VesperBat extend the fields/methods of Mammal, but in different ways

abstract keyword

Abstract Classes

- An abstract class has the keyword `abstract` in its heading
- An abstract class cannot be initialized using the `new` keyword
- An abstract class may declare abstract methods (similar to method headings in an interface)
- Any subclass of an abstract class must implement the abstract methods

Purpose of Abstract Classes

- Use an abstract class as a base class when you have two subclasses that have overlapping attributes
- Declare common fields and methods in the base class
- Declare unique fields and methods in each subclass separately

Abstract Class Example

```
public abstract class Food {
  public final String name;
  private double calories;
  // Actual methods
  public double getCalories () {
    return calories;
  }
  protected Food (String name, double calories) {
    this.name = name;
    this.calories = calories;
  }
  // Abstract methods
  public abstract double percentProtein();
  public abstract double percentFat();
  public abstract double percentCarbs();
}
```

Interfaces vs. Abstract Classes

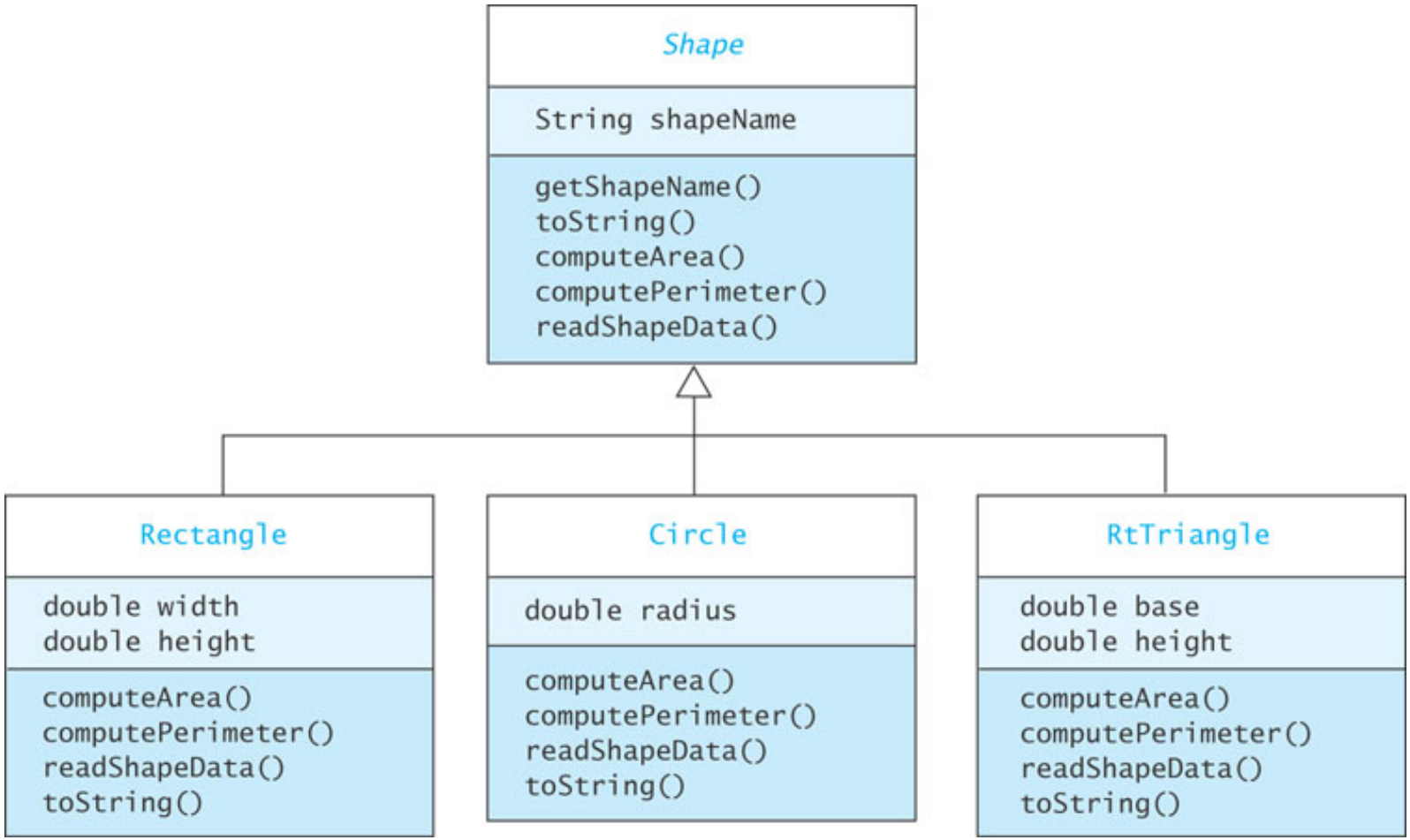
Comparison of Interface/Abstract Class

- A Java interface can declare methods but not implement them
- A Java abstract class can either implement methods or declare them `abstract`
- Interfaces can only have constant values, while classes can have variable fields
- Neither can be instantiated
- Abstract classes can have constructors defined

Usage

- Interfaces leave *all* of the implementation details to be defined in an implementing class
- Abstract classes leave *some* of the implementation details for an extending class
- A class can extend 0 or 1 superclass
- A class can implement 0 or more interfaces
- An interface cannot implement another interface but it can extend another interface (by adding more required methods)

Class Hierarchy Example



A Shape class with three subclasses that extend Shape