

Abstract Classes and Interfaces (?)

June 21, 2017

Reading Quiz

Abstract Classes

- A. Abstract classes inherit from multiple parents, standard classes only inherit from one parent.
- B. Abstract classes can only have one instance at a time, standard classes can have any number.
- C. Abstract classes cannot be directly instantiated, standard classes can be directly instantiated
- D. Abstract classes can only contain primitive properties, standard classes can have primitive and reference properties.

Abstract Methods

- A. Abstract classes consist of only abstract methods
- B. Abstract methods do not specify their implementation
- C. Abstract methods are the same thing as a "static final" method
- D. Abstract methods perform faster than standard methods

Interfaces vs. Abstract Classes

- A. Interfaces always inherit from Abstract Classes
- B. Abstract Classes always implement Interfaces
- C. Interfaces cannot inherit from Abstract Classes
- D. Abstract Classes cannot implement Interfaces

Whats the Result?

```
abstract class Person {  
    public String name() {  
        return "Person";  
    }  
}  
  
class Child extends Person {  
    public String name() {  
        return "Child";  
    }  
}  
  
Child aKindChild = new Child();  
Person aMeanPerson = new Person();  
  
System.out.println(  
    aMeanPerson.name() + "💔" +  
    aKindChild.name()  
);
```

A. "Child💔Person"

B. "Person💔Child"

C. "null💔null"

D. Won't compile

E. Runtime error

Whats the Result?

```
interface Emojier {  
    public String asEmoji();  
}  
  
class Dog implements Emojier {  
    public String asEmoji() {  
        return "🐶";  
    }  
}  
  
class Cat implements Emojier {  
    public String asEmoji() {  
        return "🐱";  
    }  
}  
  
Cat toonces = new Cat();  
String anEmoji = toonces.asEmoji();  
System.out.println(anEmoji);
```

A. "🐱"

B. "🐶"

C. "" (ie empty string)

D. Won't compile

E. Runtime error

Done!

Housekeeping

- Piazza Recap / Java environments
- Homework 2 questions
- Homework 3 incoming

Abstract Classes

Problem

- Types express a Taxonomy **and** data / functionality
- Some types are meant to represent data in a program
- Some types are "only" categorization
- Code should enforce this distinction

Animals

```
graph LR; Animals --> Mammals; Animals --> Reptiles; Mammals --> Dogs; Mammals --> Whales; Reptiles --> Snakes; Reptiles --> Lizards; Dogs --- WB1[isWarmBlooded() -> true]; Whales --- WB1; Snakes --- WB2[isWarmBlooded() -> false]; Lizards --- WB2;
```

Mammals

isWarmBlooded() -> true

Dogs

Whales

Reptiles

isWarmBlooded() -> false

Snakes

Lizards

Categorization ←

→ Program Logic

Animals

Mammals

isWarmBlooded() -> true

Dogs

Whales

Reptiles

isWarmBlooded() -> false

Snakes

Lizards

Categorization ←

→ Program Logic

Animals

Mammals

isWarmBlooded() -> true

Dogs

Whales

Reptiles

isWarmBlooded() -> false

Snakes

Lizards

Abstract

Java Solution: Abstract Classes

- Can not be instantiated, but can be inherited
- Standard inheritance model
- Arbitrarily deep

Animals

```
graph TD; Animals[Animals] --> Mammals[Mammals]; Mammals --> Whales[Whales]; Mammals --> Dogs[Dogs]; Dogs --> SadDogs[Sad Dogs]; SadDogs --> Kujo[Kujo]; SadDogs --> OldYeller[Old Yeller];
```

The diagram is a hierarchical tree structure. At the top level is a yellow rounded rectangle labeled 'Animals'. A vertical line descends from 'Animals', and a horizontal line branches off to the right, connecting to a yellow rounded rectangle labeled 'Mammals'. From 'Mammals', a vertical line descends, and two horizontal lines branch off to the right, connecting to two blue rounded rectangles: 'Whales' (top) and 'Dogs' (bottom). From 'Dogs', a vertical line descends, and a horizontal line branches off to the right, connecting to a yellow rounded rectangle labeled 'Sad Dogs'. From 'Sad Dogs', a vertical line descends, and two horizontal lines branch off to the right, connecting to two blue rounded rectangles: 'Kujo' (top) and 'Old Yeller' (bottom).

Mammals

Whales

Dogs

Sad Dogs

Kujo

Old Yeller

Animal

Mammals

Whale

Dog

SadDog

DeadDog

JerkDog

Which are Valid?

- A. new Mammal()
- B. new SadDog()
- C. new Dog()
- D. new JerkDog()

Abstract Use Cases?

	Abstract	Concrete Subtype(s)
1	SimpsonCharacter	Bart, Lisa
2	AdminUser	StandardUser, AnonymousUser
3	MembersOfTheRamones	JoeyRamone, TommyRamone
4	CreditCardChargeError	RareCreditCardChargeError

Animal.java →

Abstract Methods

- Taxonomy can impose "helpful" restrictions too
- Things below me must implement these methods
- Keep types clean

Fur.java ->

Interfaces

Problem

- Classes categorize data in a single, global way
- Multiple categorizations possible
- Redundancy, or insane hierarchies

TrickyTypes.md ->

Entertainment

```
graph LR; Entertainment --> Live; Entertainment --> Recorded; Live --> OperaPerformance[Opera Performance]; Live --> JazzPerformance[Jazz Performance]; Live --> WhirlyBall[Whirly Ball]; Recorded --> Records; Recorded --> Books; Recorded --> Photographs;
```

Live

Opera Performance

Jazz Performance

Whirly Ball

Recorded

Records

Books

Photographs

Entertainment

```
graph LR; Entertainment --> Live; Entertainment --> Recorded; Live --> OperaPerformance[Opera Performance]; Live --> JazzPerformance[Jazz Performance]; Live --> WhirlyBall[Whirly Ball]; Recorded --> Records; Recorded --> Books; Recorded --> Photographs;
```

The diagram is a hierarchical tree structure. At the top level is the node 'Entertainment'. A vertical line descends from 'Entertainment', with two horizontal lines branching off to the right, leading to the nodes 'Live' and 'Recorded'. From 'Live', another vertical line descends with three horizontal lines branching off to the right, leading to 'Opera Performance', 'Jazz Performance', and 'Whirly Ball'. From 'Recorded', a similar vertical line descends with three horizontal lines branching off to the right, leading to 'Records', 'Books', and 'Photographs'. The 'Entertainment', 'Live', and 'Recorded' nodes are blue, while 'Opera Performance', 'Jazz Performance', and 'Records' are yellow. 'Whirly Ball', 'Books', and 'Photographs' are blue.

Live

Opera Performance

Jazz Performance

Whirly Ball

Recorded

Records

Books

Photographs

Entertainment

Live

Opera Performance

Jazz Performance

Whirly Ball

Recorded

Vinyl

Books

Photographs

```
void record(? item);
```

Java Solution

- Interfaces
- Hierarchical, but separate
- Same modifiers, same inheritance rules

Interfaces vs. Classes

- Classes for data
 - Animals
 - Music Categories
 - Movies
- Interfaces for Functionality
 - Pet-able
 - Danceable
 - Recordable

Interfaces.java ->

Interfaces Specifics

- Classes can implement multiple interfaces
 - ArrayList **implements** Serializable, Iterable, Collection, List
- Small, light
- Similar to Abstract Classes
 - Inheritance, default methods



Taxonomizing File Types

- `/etc/mime.types`
- Mime is a very simple taxonomy system
- Lets implement in code!