CS151

Inheritance
Start of the Java class hierarchy

http://web.deu.edu.tr/doc/oreily/java/langref/ch10_js.htm
Java Object Methods

- `public boolean equals(Object ob)`
- `public String toString()`
- `public Class getClass()`
- `protected Object clone()`
- `protected void finalize()`
- `public int hashCode()`
- `public void notify()`
- `public void notifyAll()`
- `public void wait()`
- `public void wait(long l)`
- `public void wait(long l, int ii)`
public class Inherit extends Object {
    public static void main(String[] args) {
        Inherit inh1 = new Inherit();
        Inherit inh2 = new Inherit();
        Inherit inh3 = inh1;

        System.out.println(inh1); // implicit use of toString()
        System.out.println(inh2.toString()); // explicit toString
        System.out.println("Equals " + inh1.equals(inh2));
        System.out.println("Equals " + inh1.equals(inh3));
        System.out.println("== " + (inh1 == inh2));
        System.out.println("== " + (inh1 == inh3));
    }
}
public class Inherit2 {
    @Override
    public String toString() {
        return "Inherit2 toString " + super.toString();
    }
    @Override
    public boolean equals(Object o) {
        return this == o;
    }
    public static void main(String[] args) {
        Inherit inh1 = new Inherit();
        Inherit2 inh2 = new Inherit2();
        System.out.println(inh1);
        System.out.println(inh2);
        System.out.println("Equals " + inh1.equals(inh1));
        System.out.println("Equals " + inh2.equals(inh1));
    }
}
Overloading

```java
public class Inherit3 extends Object {
    private int value; // just hold a value from the constructor.
    public Inherit3() { this(0); }
    public Inherit3(int vvv) { this.value = vvv; }
    public boolean equals(Inherit3 o3) {
        System.out.print("I am here ");
        return o3.value == this.value;
    }
    public static void main(String[] args) {
        Inherit3 inhA = new Inherit3();
        Inherit3 inhB = new Inherit3(6);
        Inherit3 inhC = new Inherit3(6);
        System.out.println("Equals " + inhB.equals(inhA));
        System.out.println("Equals " + inhB.equals(inhC));
        System.out.println("Equals " + inhB.equals((Object) inhC));
    }
}
```
Classes and Inheritance

Consider Pets in a classic Venn Diagram view

- Fish
- Cats
- Dogs
  - Herding
  - Border Collie
  - Working
In this view, lines indicate shared traits. This is the “inheritance” in OO. Dark Blue indicates classes that should not have instances.
Pet UML

- UML is
  - “Unified Modeling Language”
  - A programming language independent way of expressing classes
  - (I will not use +/-)
public class Pet {
  protected String id;
  protected String name;
  public String sound() {
    return "silence";
  }
  public String getId() {
    return id;
  }
  public String getName() {
    return name;
  }
  public boolean equals(Pet p) {
    return id.equals(p.getId());
  }
}

public class PoorPet extends Object {
  private String id;
  private String name;
  public String sound() {
    return "silence";
  }
  public String getId() {
    return id;
  }
  public String getName() {
    return name;
  }
}

changed "private" to "protected"
added "equals"
public class Cat extends Pet {
  private String breed;
  private double hairLength;
  public Cat(String name, String id, String breed) {
    this.name = name;
    this.id = id;
    this.breed = breed;
  }
  @Override
  public String sound() {
    return "meow";
  }
  @Override
  public String toString() {
    return "My name is " + name + " breed " + breed + " and I say " + sound();
  }
  public static void main(String[] args) {
    System.out.println(new Cat("calypso", "112234", "siberian"));
  }
}
Dog Classes
Not showing constructors

```java
public class Dog extends Pet{
    protected String group;
    protected double hairLength;
    protected boolean doubleCoat;
    @Override
    public String sound() {
        return "arf";
    }
    @Override
    public String toString() {
        return sound();
    }
}
```

```java
public class WorkingDog extends Dog{
    protected String breed;
    protected String task;
    @Override
    public String toString() {
        return super.toString() + "work " + task;
    }
    @Override
    public String sound() {
        return "woof";
    }
}
```
Casting, Classes and Inheritance

• Suppose: SPCA pet shelter
• Desire: A program that tracks all animals at shelter
• Approach
  • Use single array to hold all Pets
• Complaint: Mixed the problem of storing animals with the shelter’s needs
  • better to separate the storage problem from the other needs of the shelter

• The storage problem is exactly what data structures are for

```java
public class Shelter {
    protected Pet[] animals = new Pet[100];
    protected int animalCount=0;
    public void addAnimal(Pet animal) {
        animals[animalCount++] = animal;
    }
    public Pet getAnimal(int location) {
        return animals[location];
    }
    public static void main(String[] args) {
        Shelter shelter = new Shelter();
        shelter.addAnimal(new Dog());
        shelter.addAnimal(new Cat());
    }
}
```
Data Structure for Shelter

• Desired Behaviors
  • Add an Item
  • Remove a particular item
  • Number of times a particular item appears
    • for a shelter probably should be 1, maybe CatDog should be in twice
  • Does structure contain particular item?
  • Others?

None of these reqs have anything to do with shelter. So we can make a structure to do this for shelter AND others.
**UML**

**BAG:**

- `numberOfItems`: int
- `empty`: boolean
- `add(new item)`: boolean
- `remove`: item
- `remove(an item)`: boolean
- `clear`: void
- `countOf(item)`: int
- `contains(item)`: boolean
- `display`: void
Java Interfaces

- No data fields
- No constructors
- No private methods
- No protected methods
- No bodies for methods
- Lots of instructions about how the IO behavior of methods
- I will tend to use Java interfaces rather than UML
- javadoc BagOfPets.java

```java
/**
 * Interface definition for Bag
 * Adapted slightly from Carrano & Henry
 * @author GTowell
 * Created: July 2021
 */

public interface BagOfPets {
    /**
     * The number of pets in the bag
     * @return the number of pets in the bag
     */
    public int numberOfItems();

    /**
     * true if there is at least one pet in the bag
     * @return true if there is at least one pet in
     */
    public boolean isEmpty();

    //etc
```
Java Interfaces

In a file Vehicle.java

```java
public interface Vehicle {
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

Interfaces are usually EXTENSIVELY documented so programmers know what an implementation should do. For example:
https://docs.oracle.com/javase/8/docs/api/java/

Methods defined in interfaces are always public, so public can be omitted. Clashes with class definition in which “” indicates package (Horrific inconsistency!)
Java Interfaces

• Java allows only single inheritance.
  • A class can only extend one class
    • public class Myclass extends Pet
  • Why only one?
    • Collision resolution
• BUT a class can “implement” any number of Interfaces
  • Interfaces only define methods
    • they do not provide method bodies so no collision resolution required.
    • Programmer of class that implements interface MUST write method bodies
      • resolve any issues from 'documentation collision'
Think before coding

• Point of UML (and one of the points of Java interfaces) is to get you to think about a problem before writing code

• Please do so

• While writing code,
  • get up and walk about
  • talk to a classmate about your thoughts
    • Talk to TAs about thoughts

• Start early ... please
  • early grading bonus
Implementing BagOfPets

• java
  • public X implements Y
  • This says making a class that will provide bodies for EVERY method in interface Y
  • Possibly more methods
    • private or protected helpers for public
    • private instance variables

```java
/**
 * An implementation of the BagOfPets interface
 *
 * Note that everything marked with @Override does not need documentation as it should be documented elsewhere.
 *
 */
public class PetBag implements BagOfPets {

  @Override
  public int numberOfItems() {
```
In class

• Continue implementation