Interfaces

CMSC 202

Public Interfaces

- Objects define their interaction with the outside world through the their *public interface*.
- A class' public interface is the set of public members that a user can access
  - Public non-static members & methods
  - Public static members & methods

Public Interface

- An interface is a group of related methods that multiple objects have in common, but might function slightly different.
- For instance all Vehicles
  - Speed up
  - Slow down
  - Turn

Java Interfaces

- Interfaces in Java are a set of behaviors that are common to multiple classes.
- The implementation of an Interface is similar to a class except that Interfaces:
  - Use the keyword `interface` instead of class,
  - Can only contain public methods, variables, and constants.
  - And methods are do not contain a body.
  - All methods are implicitly abstract.

Java Interface

```
public interface Drivable {
    void accelerate(int amount);
    void decelerate(int amount);
    void move(int time);
    void turn(int radians);
    double pi = 3.141;
}
```

- Each method defined in the interface does not have a body.
- Interfaces can only have initialized variables.
- Any class that has the same methods defined in the interface may *implement* Drivable.

Implementing Interfaces

```
public class Vehicle implements Drivable{
    public void accelerate(int amount){
        // accelerate like a Vehicle
    }
    public void decelerate(int amount){
        // decelerate like a Vehicle
    }
    public void turn(int radians){
        // turn like a Vehicle
    }
    public void move(int time){
        // move like a Vehicle
    }
}
```

- A class that uses an interface must
  - Use the keyword `implements`
  - And define all methods that are part of the interface.

Interfaces

- All methods are implicitly abstract.
  - A class that implements an Interface must implement all methods defined in the interface to be concrete.
  - A class that does not implement all methods must be labeled as abstract when appropriate.
- Interfaces can be used as a reference variable type.
  - All the rules of polymorphism apply.

```
Drivable thing = new Vehicle();
```

Comparable

- Comparable is an Interface defined in the Java API that is used to provide an ordering of Objects of the same type.
- A class that implements comparable must define the compareTo() method that when invoked ***a.compareTo(b)*** returns
  - -1 if a < b
  - 0 if a == b
  - 1 if a > b
- This is the convention for the compareTo method.

Comparable

```
public class Car extends Vehicle implements Comparable<Car>{
   private String make, model;

   public int compareTo(Car other){
      int result = this.make.compareTo(other.make);
      if(result < 0 || result > 0){
         return result;
      }
      else{
         return model.compareTo(other.model);
      }
   }
   public static void main(String[] args){
      Comparable car1 = new Car();
      Car car2 = new Car();
      String solution = "";
      switch(car1.compareTo(car2)){
         case -1: solution = " preceeds ";      break;
         case 0: solution = " same "; break;
         case 1: solution = " succeeds "; break;
      }
      System.out.println("car 1" + solution "car 2");
   }
}
```

- Here we are alphabetically ordering Cars by their make and then their model.

Conventions

- All interface methods must thorough javadoc comments. These must include the intended purpose of the method.
- compareTo is supposed to return specific values when invoked. This is a convention of the interface and is enforced by the Java compiler.
  - The following implementation is syntactically correct but violates the intended usage.

```
public int compareTo(Car other){
    return 0;
}
```

Design By Contract

- Design by Contract is a metaphor on how elements of a software collaborate with each other, on the basis of mutual obligations and benefits.
  - The supplier must provide a certain product (obligation) and is entitled to expect that the client has paid its fee (benefit).
  - The client must pay the fee (obligation) and is entitled to get the product (benefit).
  - Both parties must satisfy certain obligations, such as laws and regulations, applying to all contracts.

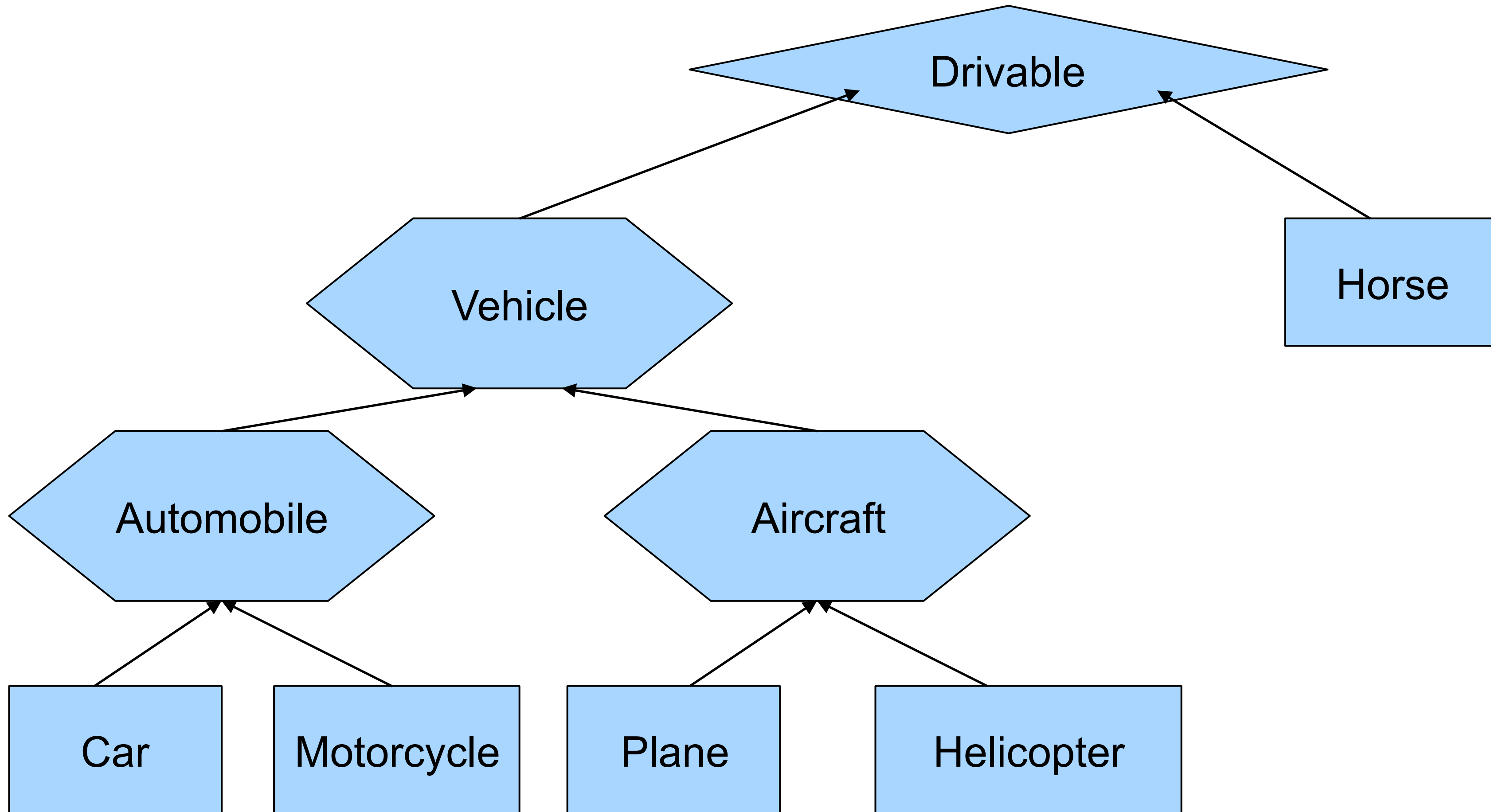
Design by Contract

- Part of design by contract is defined in the pre and post conditions defined by the supplier.
  - Preconditions – remove error checking that a user has to implement
  - Postconditions – state what is true open exit of the function guiding a user in the implementation
- The Contract is a formalization of obligations and benefits.
  - What does it expect?
  - What does it guarantee?
  - What does it maintain?

Interface Hierarchy

- Interfaces can be used as the base “class” of other interfaces
  - You can derive an interface from another interface using the keyword extends
  - A derived interface inherits all the methods of the base interface.
  - To implement a derived interface, all methods must be implemented by the class.
- Since interfaces can be used as reference variables they can add to class hierarchies

Class Hierarchy



Interfaces and Polymorphism

- We can use interfaces to increase the extensibility of our code.
  - We can write functions that require object to implement an interface instead of being a class.

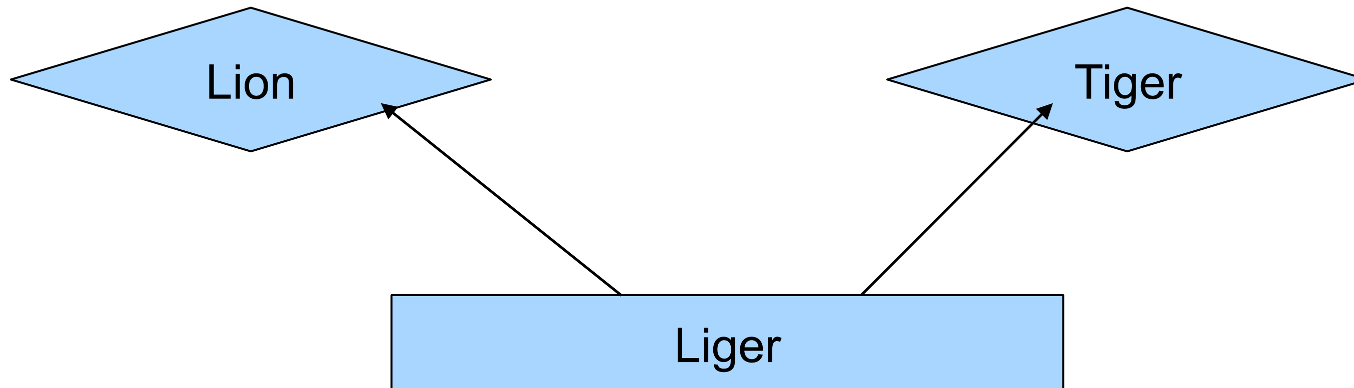
```
public static void selectionSort(Comparable<T>[] items){
   int minPos;
   int minItem;

   for(minPos = 0; minPos < items.length; minPos++){
      minItem = minPos;
      for(int i = minPos + 1; i < items.length; i++){
         if(items[i].compareTo(items[minItem]) < 0){
            // found a new minimum
            minItem = i;
         }
      }
      if(minItem != minPos){
         Comparable tmp = items[minItem];
         items[minItem] = items[minPos];
         items[minPos] = tmp;
      }
   }
}
```

Multiple Inheritance

- Java does not support multiple inheritance with classes.
  - You can not say Class X extends X, Y
- However, Java does allow a class to implement multiple interfaces.

```
public class Liger implements Tiger, Lion
```

Multiple Interfaces



- The class Liger must implement all methods in both Lion and Tiger.
- Liger can now be referenced by either Lion or Tiger but is limited to the interface defined in Lion or Tiger respectively.

Multiple Interfaces

```
public interface Lion{
   public void eat();
}
```

```
public interface Tiger{
   public void eat();
}
```

```
public class Liger implements Lion, Tiger{
   public void eat() {
      // should I eat like a lion or tiger?
   }
}
```

- No Java syntax errors will occur for methods that “overlap” from Lion and Tiger.
  - But in languages like C++ many problems arise from a “Diamond of Death”