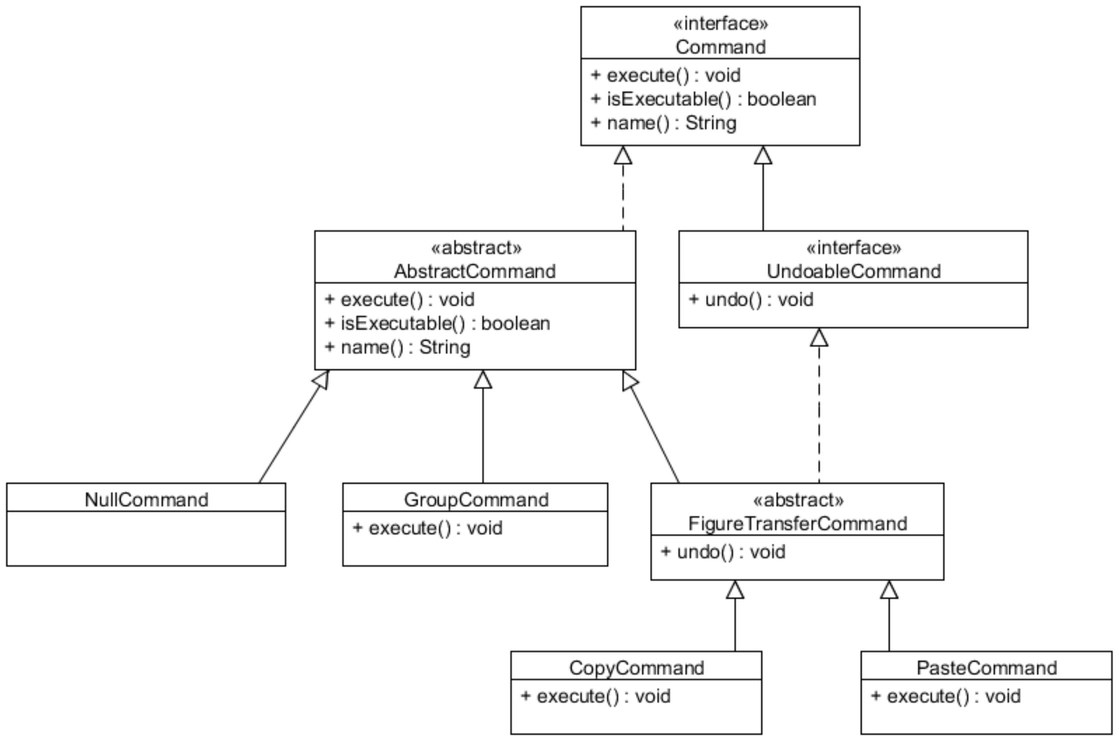
Q1) [4 marks] Type Hierarchy, Polymorphism, Dispatch:

Use the type hierarchy given below to answer the questions. Remember that if a method appears in a subclass that has the same signature as a method in one of its superclasses, that method is being overridden in the subclass.



a)	`Command c = new GroupCommand(…);`
	`c.execute();`
	
	When the above code runs, which type's execute method will be run?
	
	`GroupCommand`

b)	`AbstractCommand ac = new CopyCommand(…);`
	`ac.execute();`
	
	When the above code runs, which type's execute method will be run?
	
	`CopyCommand`

c)	`Command c = new NullCommand(…);`
	`c.execute();`
	
	When the above code runs, which type's execute method will be run?
	
	`AbstractCommand`

d)

```
FigureTransferCommand ftc = new PasteCommand(…);
AbstractCommand ac;
ac = ftc;
ac.execute();
```

When the above code runs, which type's execute method will be run?

```
PasteCommand
```

e)

```
Command c = new GroupCommand(…);
c.name();
```

Will the call to the name method compile?

```
Yes
```

If so, when the code runs, which type's name method will be executed?

```
AbstractCommand
```

f)

```
Command c = new PasteCommand(…);
c.undo();
```

Will the call to the undo method compile?

```
No
```

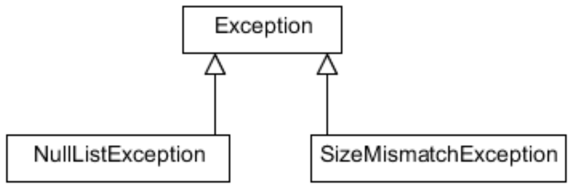
If so, when the code runs, which type's undo method will be executed?

```
N/A
```

Q2) [5 marks] Designing Robust Classes

a) **[3 marks]** Consider the `printLabSections` method given below. Redesign the method so that it is robust. You should update the specification and the implementation. Before you start, please familiarize yourself with the exception type hierarchy given below. If you think it's appropriate to use exceptions in your answer, you must use an exception type from the hierarchy given below.

```
/*
 * REQUIRES: names is not null, labs is not null and
 *           names has the same size as labs
 * EFFECTS: prints the name and lab section of each student
 */
public void printLabSections(List<String> names, List<String> labs){
      for (int i = 0; i < names.size(); i++){
            System.out.println(names.get(i) + " is in lab section " + labs.get(i));
      }
}
```



All of the exception types have two constructors – one that takes no arguments and another that takes a string that represents a message that describes the exception.

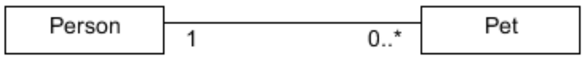
```
/**
 * EFFECTS: if names is null or labs is null, throws NullListException
 *          if size of names does not match size of labs,
 *            throws SizeMismatchException
 *          otherwise prints the name and lab section of each student
 */
public void printLabSections(List<String names, List<String> labs)
       throws NullListException, SizeMismatchException {
    if (names == null)
        throw new NullListException("list of names is null");
    if (labs == null)
        throw new NullListException("list of labs is null");
    if (names.size() != labs.size())
        throw new SizeMismatchException("size of lists not equal");

    for (int i = 0; i < names.size(); i++) {
        System.out.println(names.get(i) + " is in lab section " + labs.get(i));
}
```

b) **[2 marks]** Describe any tests that you would need to test your new robust specification. For each test, you must fill in one row of the below table. You may not need to use all the rows in the table. You can assume that there were already tests that sufficiently tested the original method and do not need to include them in your answer. You do not need to write code to describe the parameters or the expected behaviour; a clear English description is all that's needed.

Description of argument passed to `names` parameter	**Description of argument passed to `list` parameter**	**Expected behaviour**
`null`	`null`	**throws** `NullListException`
`null`	**not** `null`	**throws** `NullListException`
not `null`	`null`	**throws** `NullListException`
list having size n	**list having size other than n**	**throws** `SizeMismatchException`

Q3) [6 marks] Implementing an OO Design

Write a partial implementation for the `Person` and `Pet` classes based on the UML class diagram shown below. For each class, you must include the necessary fields along with a specification and a stub for each method that you believe is necessary to support the relationship shown on the diagram. Note that a person cannot have any duplicate pets and that once a `Pet` is adopted by a `Person`, it will always belong to that `Person`.



```
public class Person {
    private Set<Pet> pets;

    /*
     * EFFECTS: set of pets is empty
     */
    public Person() {
        pets = new HashSet<Pet>();
    }

    /*
     * MODIFIES: this
     * EFFECTS: adds p to this person's pets, if not already added
     */
    public void addPet(Pet p) {
        pets.add(p);
    }
}


public class Pet {
    private Person owner;

    /*
     * EFFECTS:
     * - sets owner of this pet to owner
     * - adds this pet to owner's pets, if not already added
     */
    public Pet(Person owner) {
        this.owner = owner;
        owner.add(this);
    }
}
```

NOTE: although not required by the question, we have included a full implementation for the methods.

Q4) [5 marks] This question relates to the AirlineInventory project checked out of the exams repository. Run the application by right-clicking:

`ca.ubc.cpsc210.inventory.app.AirlineInventoryApp.java`

and choosing Run As -> Java Application from the pop-up menu.

The application can be used by an airline to keep track of what kinds of aircraft it has in its inventory. It needs to be able to look up the aircraft by manufacturer. So, once the user specifies the manufacturer, the set of aircraft produced by that manufacturer in the airline's inventory is printed on the screen.

Start by entering "all" and notice that the inventory has already been populated with a number of aircraft. Further notice that there are duplicates – specifically, the Boeing B747 400 appears twice. This is not the desired behaviour. We want each aircraft with a given model and version number built by a particular manufacturer to appear only once in the inventory.

a) Identify the bug in the code and describe it:

   Duplicate aircraft objects can be added to `Set<Aircraft>`

b) Now describe a *minimal set* of changes necessary to fix the problem given the current bug in the application. Do not write any code. Simply write a few sentences to precisely describe what needs to be done to fix the bug:

   Override `hashCode` **and** `equals` **in** `Aircraft` **using the** `model` **and** `version` **fields, so that duplicate** `Aircraft` **objects cannot be added to** `Set<Aircraft>`**.**

Q5) [6 marks] This question relates to the TextParserMT2 project checked out of the exams repository.

Add a class `SmileyFaceParser` that extends the `AbstractTextParser` class. You must override the `parse` method so that it parses all instances of the `String` `":)"` found in the string to parse that's passed as an argument when the method is called. You might find it helpful to study the parse method in the `AbstractTextParser` class as some aspects of its implementation are similar to the method that you will write in this question.

Note that the project includes a JUnit test class `TestSmileyFaceParser`. If your implementation is correct, these tests will pass. However, note that the fact that the tests pass is not a guarantee that your implementation is correct.

```
public class SmileyFaceParser extends AbstractTextParser {

    @Override
    public boolean parse(String toParse) {
        final int ICONSIZE = 2;
        final String SMILEY = ":)";
        // Represents the position in toParse from which we are
        // attempting to extract a smiley face.
        int curPos = 0;
        // Cache the length of toParse.
        int len = toParse.length();

        while (curPos < len) {
            // smileyIndex holds the position of the next
            // occurrence of ":)" in toParse, starting from
            // position curPos.
            int smileyIndex = toParse.indexOf(SMILEY, curPos);

            if (smileyIndex == -1) {
                break;
            }

            // Extract the next smiley face.
            String itemString
                   = toParse.substring(smileyIndex,
                                       smileyIndex + ICONSIZE);

            items.add(itemString);

            curPos = smileyIndex + ICONSIZE;
        }

        return true;
     }
}
```