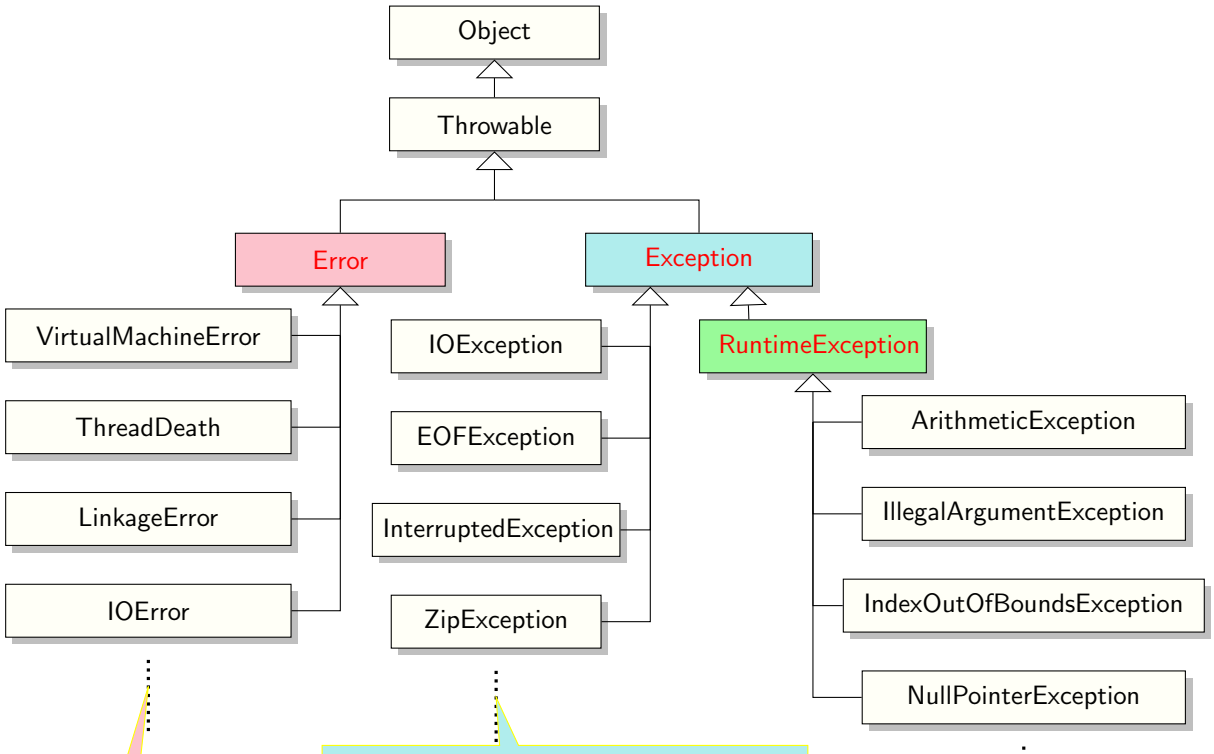
420-201-VA | Lecture + Last Assignment 4-B | Due Date: 17 April 2020

Basic Java Exception Handling

1 Purpose

- To learn and practice exception handling in java

2 Part of the Java Error and Exception class hierarchy



Unchecked, all have `Error` class in their inheritance hierarchy, and all indicate fatal &unrecoverable JVM errors. We should ignore these errors.

Checked exceptions

None has `RuntimeException` but all have `Exception` class in their inheritance hierarchy.

Java forces you to either declare or use `try` and `catch` mechanism to handle a checked exception.

A checked exception is thrown to indicate that an error outside the control of our program has occurred.

Unchecked exceptions

All have `RuntimeException` class in their inheritance hierarchy. Java does *not* force you to either declare or use `try` and `catch` to handle an unchecked exception, but if you do, then you handle an unchecked exception similar to a checked exception.

An unchecked exception is thrown to indicate there is a logical error in our program, pointing the responsibility finger at us, the programmer.

3 Exception Handling Mechanism

An exception in Java is an object that represents an abnormal processing situation that occurs during the program execution, typically storing a message about the cause of the exception and potentially storing other information about the abnormal situation.

An exception halts the normal execution of the method in which it occurs, providing an opportunity for the method to handle the exception.

The exception handling mechanism in Java involves three parts:

3.1 Defining an Exception

Suppose we are processing a comma-separated values (CSV) file in which each line is supposed to contain data about only three types of pets, namely, dog, cat, and duck; for example:

```
dog  , German Shepherd, Tux,10 , f
Cat , no , FluffyWhiskers ,9,  m
pig ,   1  , Charlie ,  5 ,f
```

The first value in each line represents a pet type and the second value is specific to that type, such as `breed` for dog, `yes` or `no` for cat being neutered, or an integer for duck eggs. The last three values on each line represent the name, age, and gender of a pet, in that order.

As you can see, the last line represents an unknown pet type "pig". To deal with such situation, we first need to create an object to represent the situation:

```java
public class PetTypeUnknownException extends Exception
{
   // the exception supperclass already manages a detail message
   // on the cause of this exception;

   // we can define data memebrs if we need to store information
   // other than a detail message of cause;

   // typically, in simple cases such as this one, we need no data memebrs

   // we normally provide default constructor, customizing super's detail message;
   // to do that, we call super's default constructor passing it a fixed message
   public PetTypeUnknownException()
   {
      super("Encountered an unknow pet type"); // our fixed custom message
   }
   // we also normally provide a constructor that takes a String as only parameter,
   // allowing the user to provide its own detail message;
   // to do that, we call another super's constructor passing it the user supplied message
   public PetTypeUnknownException(String user_message)
   {
      super(user_message);
   }
}
```

3.2 Throwing an Exception

An exception is thrown by a `throw` statement.

Our `PetTypeUnknownException` class defines two constructors, providing two ways to instantiate it on a `throw` statement.

Way 1

```
// assume some_undesirable_condition is a properly defined boolean variable
if(some_undesirable_condition)
{
   throw new PetTypeUnknownException();
}
```

Way 2

```
// assume some_undesirable_condition is a properly defined boolean variable
if(some_undesirable_condition)
{
   throw new PetTypeUnknownException("Error: encounterd an unkown pet type on line 3");
}
```

Question: what's the difference between `throw` and `throws` in Java?

Answer:

- The `throw` keyword is used in a method body to throw an exception and is always follows by an object of the exception class being thrown.
- The `throws` clause is used in method *header* to declare an exception and is always followed by (a list of comma-separated) exception class names.

3.3 Handling an exception

We have two options to handle (deal with) an exceptional situation in Java:

Option 1: Pass the Buck Simply do not handle the exception in the method where it occurs, passing the buck to the calling method (that is, the method that called it.)

We used this option when we processed a text file in assignment 3A, where we appended a `throws` clause to our method headers (see assignment 3A, page 4, lines 6 and 23.)

```
public static void main(String[] args) throws FileNotFoundException,
                                               PetTypeUnknownException
{
  // here our method main declares that the calling method is responsible for
  // dealing with any FileNotFoundException or PetTypeUnknownException objects
  // our main method may throw;
}
```

Option 2: Handle the Exception

```
try
{
   // ...
   // Statements that try to open a file
   // ...
   // Statements that try to validate an input line
   // ...
} catch (IOException ioe)
{
   //...
   // Statements that execute if the file can't be opened for some reason
   //...
}
catch (PetTypeUnknownException ptue)
{
   //...
   // Statements that execute if an unkown pet type encountered
   //...
}
finally
{
   //...
   // Statements that are always executed;
   // typically used to clean up resources such as closing files
   //...
}
```

Propagation of an Exception:

if an exception is not caught and handled in a method where it occurs, it is propagated to the calling method;

if the exception is not caught and handled in the calling method, it is propagated to the method that called the calling method;

and so on.

The exception continues to propagate up the calling method chain until a method in the chain handles it, or until it reaches the JVM, which, in turn, prints some messages, providing a method call stack trace (a series of method calls that led into the method in which the exception originated.)

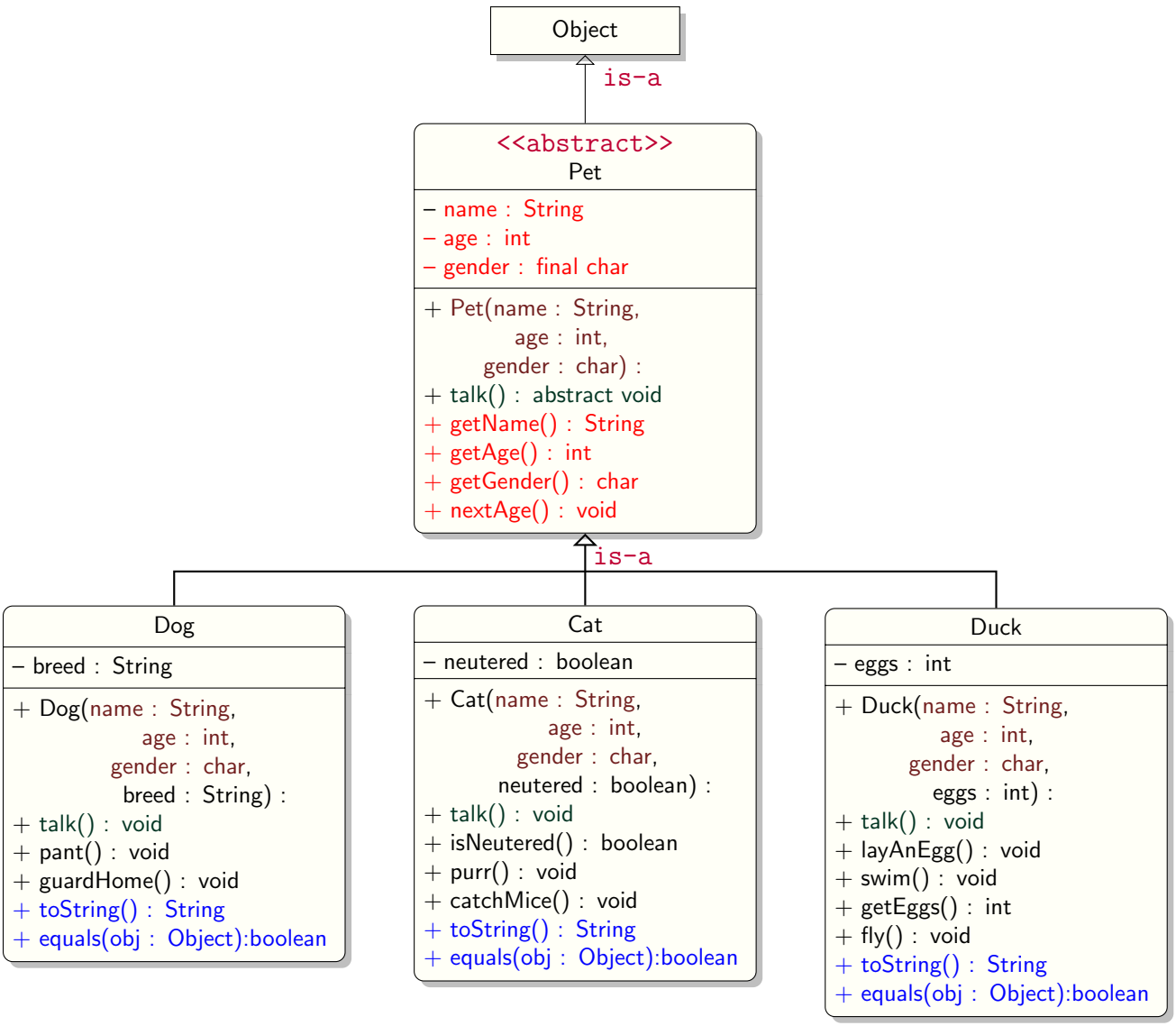
Remember:

- If a *checked* exception is not caught and handled in the method where it occurs, then the `throws` clause must be included on the method header.

4 Your Task in 10 Easy Steps

Step 1. Prepare Your Existing Pet Inheritance Hierarchy

Make copy of your project folder for assignment 4A, ensuring that it implements the UML class diagram shown on page 10 of 4A, which is reproduced[1] here for your convenience.

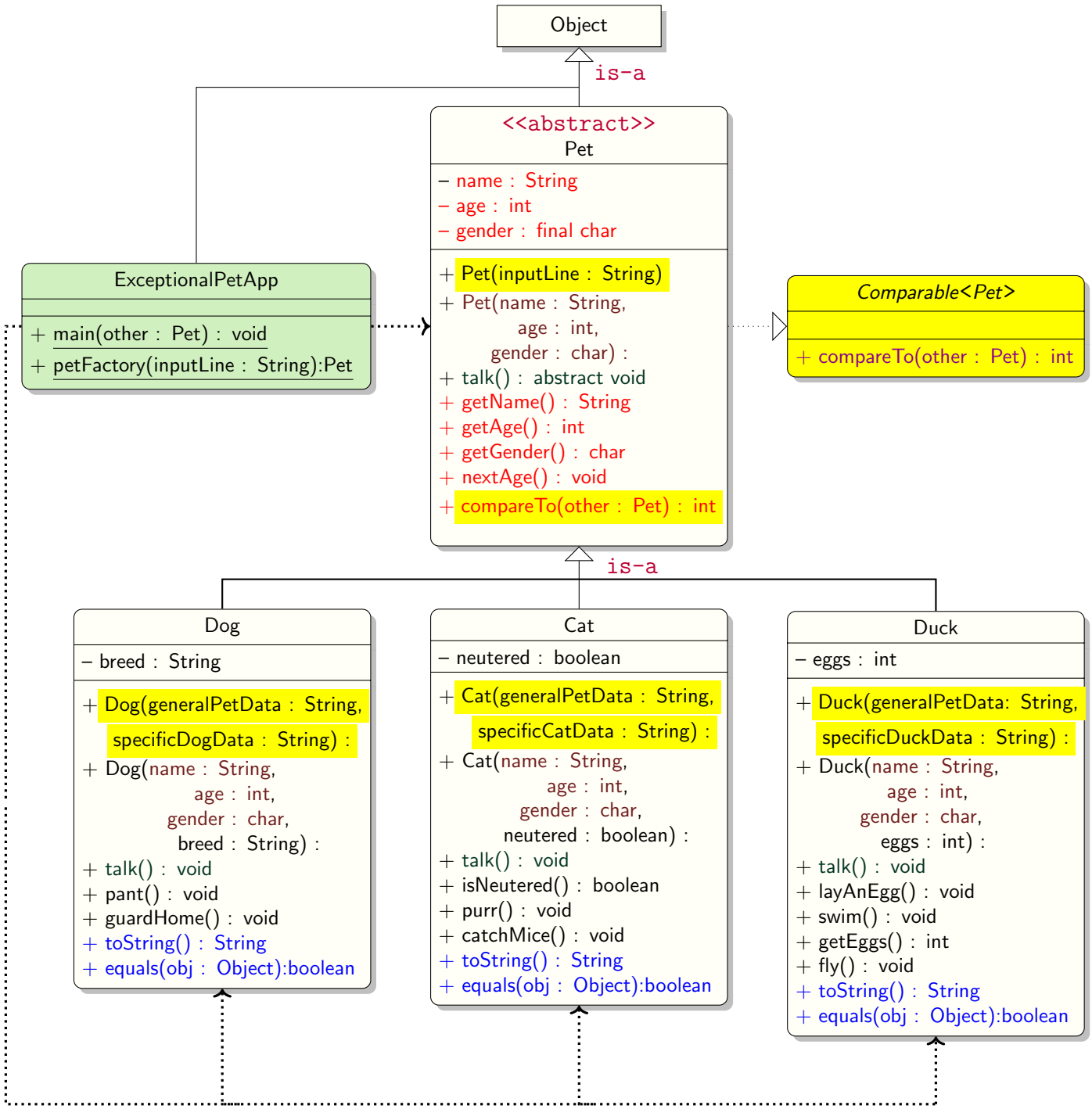


[1] without the callouts.

Step 2. Look at the big picture and follow the given steps to get there

Our goal is to enhance 4A to include four new constructors and to implement the `Comparable<Pet>` interface, which are shown in yellow in the following inheritance hierarchy.

The complete code for class ExceptionalPetApp is given with this assignment.



Step 3. Add a New Pet Constructor

Enhance your Pet class to include a new constructor `public Pet(String inputLine)` (same as lines 14-28 minus line 26 from assignment 3A, page 3:)

```
public abstract class Pet implements  Comparable<Pet>
{
   // Initializes this new pet dog object, using data values from inputline
   public Pet(String inputLine)
   {
      // create a Scanner using inputLine
      Scanner lineScanner = new Scanner(inputLine);
      // Tell lineScanner that inputLine consists of data separated by commas,
      // which are each preceded and followed by zero or more spaces
      lineScanner.useDelimiter("\\s*,\\s*");
      // extract the data values from the input line
      this.name   = lineScanner.next();    // read name
      this.age    = lineScanner.nextInt(); // read age
      this.gender = lineScanner.next().toUpperCase().charAt(0); // read gender
      lineScanner.close(); // close lineScanner
   }
   // other code not shown for brevity
   // ...
}
```

Step 4. Have Class Pet Implement the Comparator Interface

Enhance your Pet class to implement the `Comparator` interface (same as lines 3-22, assignment 3B, page 6.)

```
public abstract class Pet implements  Comparable<Pet>
{
   @Override
   public int compareTo(Pet other)
   {
       // order this and other pet objects by name and by age for equal names
       int result = this.getName().compareTo(other.getName());

      // if the names are not the same return result
      if( result != 0) return result;
      // names are the same, so sort by age
      return this.getAge() - other.getAge(); // return a negative, zero, or positive integer
   }
   // other code not shown for brevity
   // ...
}
```

Step 5. Add a New Constructor to Class Dog

Enhance the Dog class to include the following constructor:

```
public class Dog extends Pet
{

   public Dog(String petGeneralData, String petSpecificData)
   {
      super(petGeneralData); // let superclass Pet initialize the pet properties
      this.breed = petSpecificData; // let subclass Dog initialize Dog properties
   }
   // other code not shown for brevity
   // ...
}
```

Step 6. Add a New Constructor to Class Cat

Enhance the Cat class to include the following constructor:

```
public class Cat  extends Pet
{

   public Cat(String petGeneralData, String petSpecificData)
   {
      super(petGeneralData); // let superclass Pet initialize the pet properties

      // let subclass Cat initialize Dog properties
      // if the supplied neutered value is anything other than "yes" or "no"
      // throw a IllegalArgumentException, but let the calling code handle it;
      // since our IllegalArgumentException is unchecked we don't need to
      // append a throws clause to the constructor header.

      if( !(petSpecificData.equalsIgnoreCase("yes") ||
            petSpecificData.equalsIgnoreCase("no")  ))
      {
          throw new IllegalArgumentException("Bad neutered value "
                                             + petSpecificData
                                             + ", expected yes or no");
      }
      // we make it here only if petSpecificData is either "yes" or "no"
      this.neutered = petSpecificData.equalsIgnoreCase("yes");
   }
   // other code not shown for brevity
   // ...
}
```

Step 7. Add a New Constructor to Class Duck

Enhance the Duck class to include the following constructor:

```
public class Duck  extends Pet
{
   public Duck(String petGeneralData, String petSpecificData)
   {
      super(petGeneralData); // let superclass Pet initialize the pet properties

      // let subclass Duck initialize Dog properties
      this.eggs = Integer.parseInt(petSpecificData); // convert string to int

      // note: if petSpecificData does not contain a parsable integer,
      // then parseInt will throw a NumberFormatException.
      // we choose to let the calling code deal with NumberFormatException;
      // since NumberFormatException is unchecked, we don't need to add the
      // throws clause "throws NumberFormatException" to the constructor header

   }
   // other code not shown for brevity
   // ...
}
```

Step 8. Create Our Exception Class

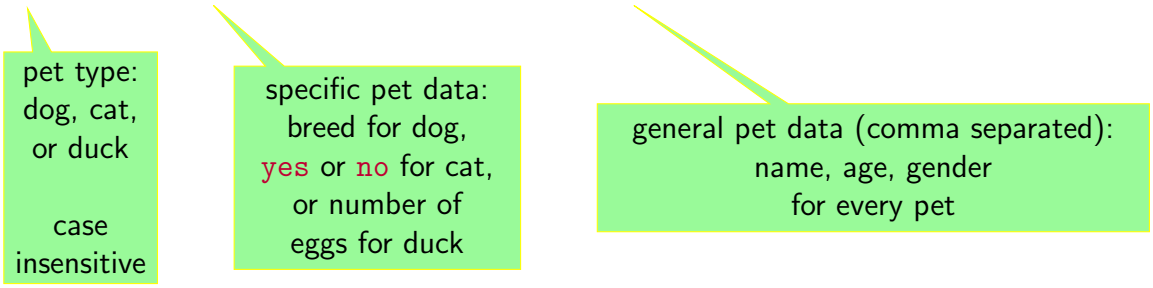
Create the following class PetTypeUnknownException, which is also shown on page 2.

```
25 public class PetTypeUnknownException extends Exception
26 {
27    public PetTypeUnknownException()
28    {
29       super("Encountered an unknow pet type"); // our fixed custom message
30    }
31
32    public PetTypeUnknownException(String user_message)
33    {
34       super(user_message);
35    }
36 }
```

Step 9. Prepare the Input File

Copy the attached `pet_infile.txt` to your project folder. Here is a sample content of the input file:

```
dog   , German Shepherd   , Tux,10 , f
Cat   ,       no          , FluffyWhiskers ,9, m
Duck ,       1            , Charlie , 5 ,f
```



There are 53 lines in the input file. The following are the top 3 lines which are all invalid:

```
1 pig ,  5, Charlie , 2,f
2 Duck, 10eggs   ,  Daffy Duck    ,2, F
3 cat   , maybe   ,  BuddyBaby, 1,  m
```

- Line 1 is invalid because "pig" is an unknown pet type
- Line 2 is invalid because the string "10eggs" does not contain a parsable integer
- Line 3 is invalid because "maybe" is not a "yes" or "no"

Step 10. Test Drive Our Exceptional Pets

The code for this driver program is bundled in a folder with this assignment.

```
public class ExceptionApplication
{    /**
     * Demonstrates exception handling of both checked and unchecked types;
     * extracts the contents of a given CSV input file line by line,
     * dissecting each line into appropriate values, creating a pet object
     * using the values, and storing the resulting pet object in an array list;
     * Demonstrate polymorphism by printing out the pet objects in the array list.
     *
     * @param args Unused.
     */
    public static void main(String[] args)// notice no throws-statement here
                                          // for UnknowPetTypeException, even
                                          // though it is checked (why?)
    {
        // normally, we don't hard code the file name; we do it here for simplicity
        String infileName = "pet_input_file.txt";
        File file = new File(infileName);
        Scanner fileScanner = null;
        try
        {   // the folowing call can potentinally throw a FileNotFoundException
            fileScanner = new Scanner(file); // try openning file for input
        }
        catch (FileNotFoundException ex)
        {
            System.out.println("Could not open the supplied input file: " + infileName);
            System.out.println("Try again later with a text file named " + infileName);
            System.out.println(" in the project folder of this project.");
            return;
        }
        // input file ok and ready to process
        // so, let's first allocate storage for our pet objects
        ArrayList<Pet> petList = new ArrayList<>();
        int lineCounter = 0;
```

```
34          while (fileScanner.hasNextLine()) // while there is a next line in the file
35          {
36              ++lineCounter;
37              String line = fileScanner.nextLine();     // read a line
38              Pet pet = null;
39              try
40              {
41  //      petFactory can potentially throw one of three exceptions (see catch blocks below)
42                  pet = petFactory(line);
43  //      if petFactorythrows an exception, the assignement operatiorn is not performed
44  //      leaving pet equal to null
45              }
46              catch (NumberFormatException ex)
47              {
48                  System.err.println(ex.getMessage());
49                  System.err.println("the string representing the number of eggs "
50                    + "does not contain a parsable integer.");
51                  System.err.println("Bad input line: " + line);
52                  System.err.println("Bad input line number: " + lineCounter);
53                  System.err.println("Bad input line ignored\n");
54                  //pet remains null
55              }
56              catch (UnknowPetTypeException ex)
57              {
58                  System.err.println(ex.getMessage());
59                  System.err.println("Pet type can be one of dog, cat or duck, "+
60                                     "case insensitive");
61                  System.err.println("Bad input line: " + line);
62                  System.err.println("Bad input line number: " + lineCounter);
63                  System.err.println("Bad input line ignored\n");
64                  //pet remains null
65              }
66              catch (IllegalArgumentException ex)
67              {
68                  System.err.println(ex.getMessage());
69                  System.err.println("The neutered value must be yes or no");
70                  System.err.println("Bad input line: " + line);
71                  System.err.println("Bad input line number: " + lineCounter);
72                  System.err.println("Bad input line ignored\n");
73                  //pet remains null
74              }
75              if( pet != null) // if no exception was thrown
76              {
77                  petList.add(pet);
78              }
79          } // end while;  Completed processing pet records
```

```
        fileScanner.close(); // remember to close files when your are done with them
        System.out.println("Completed processing " + lineCounter + " pet records");

        // show off polymorphism
        System.out.println("\nUnsorted list of pets");
        for (Pet pet : petList)
        {
            System.out.println(pet);
            // at runtime, Dog's toString() is called if pet references a Dog object
            // at runtime, Cat's toString() is called if pet references a Cat object
            // at runtime, Duck's toString() is called if pet references a Duck object
        }

        // sort using our Pet's compareTo() method for ordering of pet objects
        Collections.sort(petList);

        System.out.println("\nSorted list of pets");
        // show off polymorphism, again
        for (Pet pet : petList)
        {
            System.out.println(pet);
            // at runtime, Dog's toString() is called if pet references a Dog object
            // at runtime, Cat's toString() is called if pet references a Cat object
            // at runtime, Duck's toString() is called if pet references a Duck object
        }
    }
```

```
    /**
    * Extracts the comma separated values (CSV) from the supplied inputLine,
    * creates a Pet object accordingly, and then returns that object.
    *
    * @param inputLine The given input line.
    * @param lineCounter The line number associated with the given input line.
    * @return A Pet object of type Dog, Cat, or Duck.
    * @throws UnknowPetTypeException If input line has unknown pet type.
    * @throws NumberFormatException If number of eggs is not an integer.
    * @throws IllegalArgumentException If neutered value is neither "yes" nor "no"
    */
```

```
117     public static Pet petFactory(String inputLine) throws
118     UnknowPetTypeException   // checked, must be declared because
119                              // we choose not to handle it in this method
120 //           ,NumberFormatException,   // unchecked, no need to declare it
121 //           ,IllegalArgumentException // unchecked, no need to declare it
122    {
123       // create a Scanner using inputLine
124       Scanner lineScanner = new Scanner(inputLine);
125       // Tell lineScanner that inputLine consists of data separated by commas,
126       // which are each preceded and followed by zero or more spaces
127       lineScanner.useDelimiter("\\s*,\\s*");
128
129       // read the pet type
130       String petType = lineScanner.next();
131       // read pet specific data: breed for dog, neutered for cat, or eggs for duck
132       String petSpecificData = lineScanner.next();
133       // read pet general data which is comma separated of the form "name, age, gender"
134       String petGeneralData = lineScanner.nextLine();
135       lineScanner.close(); // close lineScanner
136
137       // next validiate the pet type
138       if (petType.equalsIgnoreCase("dog"))
139       {
140          return new Dog(petGeneralData, petSpecificData);
141       }
142       else if (petType.equalsIgnoreCase("cat"))
143       {
144          return new Cat(petGeneralData, petSpecificData);
145       }
146       else if (petType.equalsIgnoreCase("duck"))
147       {
148          return new Duck(petGeneralData, petSpecificData);
149       }
150       else
151       {  // bad pet type
152          throw new UnknowPetTypeException("\nInput error: unknown pet type \""
153                                           + petType + "\"");
154       }
155    }
156 }
```

4.1 Program Output

```
Input error: unknown pet type "pig"
Pet type can be one of dog, cat or duck, case insensitive
Bad input line: pig ,  5, Charlie , 2,f
Bad input line number: 1
Bad input line ignored

For input string: "10eggs"
the string representing the number of eggs does not contain a parsable integer.
Bad input line: Duck, 10eggs  ,  Daffy Duck   ,2, F
Bad input line number: 2
Bad input line ignored

Input error: invalid neutered value "maybe", expected yes or no
The neutered value must be yes or no
Bad input line: cat  , maybe  ,  BuddyBaby, 1,  m
Bad input line number: 3
Bad input line ignored
```

```

Completed processing 53 pet records
```

```

Unsorted list of pets
Tux, a 10 year old female pet German Shepherd dog
FluffyWhiskers, a 4 year old male pet neutered cat
Fluffy, a 5 year old female pet duck with 2 eggs
SylvesterRocky, a 5 year old male pet not neutered cat
Sparkles, a 3 year old female pet duck with 3 eggs
Lola, a 4 year old female pet Boxer dog
Duck Norris, a 6 year old male pet duck with 2 eggs
Charlie, a 1 year old male pet neutered cat
Sparkles, a 3 year old female pet duck with 5 eggs
Jack, a 9 year old male pet Miniature Schnauzer dog
Duke, a 8 year old male pet Pug dog
Lucky Ducky, a 0 year old male pet duck with 1 eggs
Molly, a 0 year old female pet Australian Kelpie dog
Toby, a 8 year old male pet Pug dog
Firequacker, a 2 year old female pet duck with 0 eggs
Rosie, a 2 year old female pet French Bulldog dog
Quackula, a 12 year old female pet duck with 0 eggs
Firequacker, a 0 year old female pet duck with 0 eggs
Ollie, a 7 year old male pet Miniature Schnauzer dog
KittySassy, a 6 year old female pet neutered cat
Dewey, a 5 year old male pet duck with 0 eggs
```

```
Huey, a 11 year old male pet duck with 2 eggs
Teddy, a 12 year old male pet Bulldog dog
Sam, a 12 year old male pet not neutered cat
Mock Duck, a 7 year old male pet duck with 5 eggs
Eggbert, a 3 year old female pet duck with 4 eggs
Eggbert, a 6 year old female pet duck with 6 eggs
Luna, a 1 year old female pet Alaskan Malamute dog
MaxSmokey, a 6 year old male pet neutered cat
Alex, a 4 year old male pet not neutered cat
Buddy, a 6 year old male pet Pug dog
Bella, a 11 year old female pet Cavalier King Charles Spaniel dog
Pepper, a 9 year old male pet neutered cat
Quack Sparrow, a 0 year old male pet duck with 2 eggs
Toby, a 4 year old male pet neutered cat
Bailey, a 11 year old female pet Australian Shepherd dog
Count Ducula, a 0 year old female pet duck with 6 eggs
Lily, a 4 year old female pet Labrador Retriever dog
Daisy, a 6 year old female pet Keeshond dog
Squeek, a 3 year old male pet duck with 6 eggs
Daisy, a 2 year old female pet Miniature Schnauzer dog
SpikeSophie, a 12 year old female pet neutered cat
Scrooge McDuck, a 9 year old male pet duck with 2 eggs
Shadow, a 8 year old female pet not neutered cat
Teddy, a 3 year old male pet Pug dog
MaggieCallie, a 10 year old female pet neutered cat
Wiggles, a 10 year old female pet duck with 3 eggs
Ollie, a 4 year old male pet Alaskan Malamute dog
Nibbles, a 3 year old female pet duck with 6 eggs
Coco, a 7 year old female pet Jack Russell Terrier dog
```

```

Sorted list of pets
```

```
Alex, a 4 year old male pet not neutered cat
Bailey, a 11 year old female pet Australian Shepherd dog
Bella, a 11 year old female pet Cavalier King Charles Spaniel dog
Buddy, a 6 year old male pet Pug dog
Charlie, a 1 year old male pet neutered cat
Coco, a 7 year old female pet Jack Russell Terrier dog
Count Ducula, a 0 year old female pet duck with 6 eggs
Daisy, a 2 year old female pet Miniature Schnauzer dog
Daisy, a 6 year old female pet Keeshond dog
Dewey, a 5 year old male pet duck with 0 eggs
Duck Norris, a 6 year old male pet duck with 2 eggs
Duke, a 8 year old male pet Pug dog
Eggbert, a 3 year old female pet duck with 4 eggs
```

```
Eggbert, a 6 year old female pet duck with 6 eggs
Firequacker, a 0 year old female pet duck with 0 eggs
Firequacker, a 2 year old female pet duck with 0 eggs
Fluffy, a 5 year old female pet duck with 2 eggs
FluffyWhiskers, a 4 year old male pet neutered cat
Huey, a 11 year old male pet duck with 2 eggs
Jack, a 9 year old male pet Miniature Schnauzer dog
KittySassy, a 6 year old female pet neutered cat
Lily, a 4 year old female pet Labrador Retriever dog
Lola, a 4 year old female pet Boxer dog
Lucky Ducky, a 0 year old male pet duck with 1 eggs
Luna, a 1 year old female pet Alaskan Malamute dog
MaggieCallie, a 10 year old female pet neutered cat
MaxSmokey, a 6 year old male pet neutered cat
Mock Duck, a 7 year old male pet duck with 5 eggs
Molly, a 0 year old female pet Australian Kelpie dog
Nibbles, a 3 year old female pet duck with 6 eggs
Ollie, a 4 year old male pet Alaskan Malamute dog
Ollie, a 7 year old male pet Miniature Schnauzer dog
Pepper, a 9 year old male pet neutered cat
Quack Sparrow, a 0 year old male pet duck with 2 eggs
Quackula, a 12 year old female pet duck with 0 eggs
Rosie, a 2 year old female pet French Bulldog dog
Sam, a 12 year old male pet not neutered cat
Scrooge McDuck, a 9 year old male pet duck with 2 eggs
Shadow, a 8 year old female pet not neutered cat
Sparkles, a 3 year old female pet duck with 3 eggs
Sparkles, a 3 year old female pet duck with 5 eggs
SpikeSophie, a 12 year old female pet neutered cat
Squeek, a 3 year old male pet duck with 6 eggs
SylvesterRocky, a 5 year old male pet not neutered cat
Teddy, a 3 year old male pet Pug dog
Teddy, a 12 year old male pet Bulldog dog
Toby, a 4 year old male pet neutered cat
Toby, a 8 year old male pet Pug dog
Tux, a 10 year old female pet German Shepherd dog
Wiggles, a 10 year old female pet duck with 3 eggs
```

5 Evaluation Criteria

Evaluation Criteria

Functionality	Ability to perform as required, producing correct output for any set of input data, Proper implementation of all specified requirements, Efficiency	60%
Robustness	Ability to handle input data of wrong type or invalid value	10%
OOP style	Encapsulating only the necessary data inside objects, Information hiding, Proper use of Java constructs and facilities.	10%
Documentation	Description of purpose of program, Javadoc comment style for all methods and fields, comments on non-trivial steps in all methods	10%
Presentation	Format, clarity, completeness of output, user friendly interface	5%
Code readability	Meaningful identifiers, indentation, spacing, localizing variables	5%