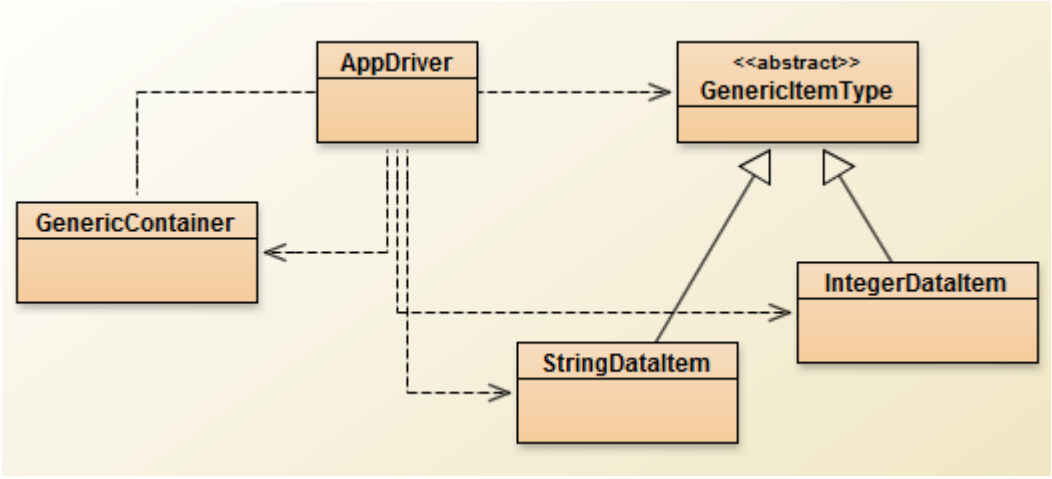
```
CISP 401
Generic Sort Application

/* Class diagram and relations for the model implemented via code below:

          GenericItemType // is an abstract class
            (+) abstract boolean isLess(GenericItemType)
            (+) abstract boolean isEqual(GenericItemType)
            (+) abstract boolean isGreater(GenericItemType)

          IntegerDataType --- 1 :  1 (inherits) ---> GenericItemType
            (+) all constructors
            (+) boolean isLess(GenericItemType) // overrides of base method
            (+) boolean isEqual(GenericItemType)
            (+) boolean isGreater(GenericItemType)
            (+) accessors (get(), toString())
            (+) manipulators

          StringDataType --- 1 :  1 (inherits) ---> GenericItemType
            (+) boolean isLess(GenericItemType) // override of base method
            (+) boolean isEqual(GenericItemType)
            (+) boolean isGreater(GenericItemType)

          GenericContainer --- 1 :  m (contains) --- GenericItemType
*/
```



```
public class AppDriver
{
public static void main(String[] args)
{
    GenericContainer gC = new GenericContainer();

    gC.add(new IntegerDataItem(13));
    gC.add(new IntegerDataItem(-30));
    gC.add(new IntegerDataItem(100));
    gC.add(new IntegerDataItem(70));
    gC.add(new IntegerDataItem(45));
    gC.sort();
    System.out.printf("     Sorted Integer Collection\n");
    gC.Iterator_Initialize();
    while (gC.Iterator_hasNext())  {
       IntegerDataItem nextOne = (IntegerDataItem )(gC.Iterator_getNext());
       System.out.printf("  %5d", nextOne.get());
       if (!(gC.Iterator_hasNext())) System.out.printf("\n\n");
    }
    GenericContainer sgC= new GenericContainer();
    sgC.add(new StringDataItem("johnson"));
    sgC.add(new StringDataItem("dixon"));
    sgC.add(new StringDataItem("adams"));
    sgC.add(new StringDataItem("Baker"));
    sgC.add(new StringDataItem("Lee"));
    sgC.add(new StringDataItem("Camille"));
    sgC.sort();
    System.out.printf("     Sorted string Collection\n\n");
    sgC.Iterator_Initialize();
    while (sgC.Iterator_hasNext())  {
       StringDataItem nextOne = (StringDataItem) (sgC.Iterator_getNext());
       System.out.printf("  %s", nextOne.get());
       if (!(sgC.Iterator_hasNext())) System.out.printf("\n");
    }
} // main
} // class
```

```
public class GenericContainer
{
      public GenericContainer()
        {sizeLIMIT=30; entriesCount=0; collection = new GenericItemType[MAXSIZE];}
      public GenericContainer(short size)
        { entriesCount = 0;
          if (size <= MAXSIZE)
             sizeLIMIT = size;
           else
             sizeLIMIT = MAXSIZE;
        }
      public GenericContainer(GenericContainer gc)
        { entriesCount = 0;
          /* Shallow COPY: collection =  gc; */
          gc.Iterator_Initialize(); // Deep COPY
          while (gc.Iterator_hasNext()) collection[inDEX]=gc.Iterator_getNext();
        }
      public void init()
        {    Iterator_Initialize();
             while (Iterator_hasNext()) collection[inDEX]= null;
        }
      public void add(GenericItemType it){collection[entriesCount++]=it;}
      public void remove(GenericItemType it){}
      public void search(GenericItemType it){}
      public void sort()
        { // bubble sort algorithm
             short outer,inner;

             for (outer=0;outer < entriesCount;outer++)
               for(inner=0; inner < entriesCount-1;inner++)
               {
                   if (collection[inner].isGreater(collection[inner+1]))
                   {
                       GenericItemType temp = collection[inner];
                       collection[inner] = collection[inner+1];
                       collection[inner+1] = temp;
                    }
                } // inner
        }
        public void Iterator_Initialize() {inDEX = 0;}
        public boolean Iterator_hasNext() {return inDEX <= entriesCount-1;}
        public GenericItemType Iterator_getNext(){ return collection[inDEX++];}
        private final int MAXSIZE = 30;
        private short sizeLIMIT,inDEX,entriesCount;
        private GenericItemType[] collection;
}
```

```
public abstract class GenericItemType
{ // class with at least one(1) abstract method is abstract class
  // an abstract method cannot contain a method body

   public abstract boolean isLess(GenericItemType git);
   public abstract boolean isEqual(GenericItemType git);
   public abstract boolean isGreater(GenericItemType git);
}

public class StringDataItem extends GenericItemType
{

   public StringDataItem(){ privateString = new String("");}
   public StringDataItem(String s){ privateString = new String(s);}
   public StringDataItem(StringDataItem sdi){ set(sdi.get());}
   public void set(String s) { privateString = s;}
   public boolean isLess(GenericItemType git) //upcasts required in each subtype
     { return ( privateString.compareTo(((StringDataItem) git).get()) < 0);}
   public boolean isEqual(GenericItemType git)
     { return ( privateString.compareTo(((StringDataItem) git).get()) == 0);}
   public boolean isGreater(GenericItemType git)
     { return ( privateString.compareTo(((StringDataItem) git).get()) > 0);}
   public String get() { return privateString;}

   private String privateString;
}

public class IntegerDataItem extends GenericItemType
{
    public IntegerDataItem(){ privateValue=0;}
    public IntegerDataItem(int i){ set(i);}
    public IntegerDataItem(IntegerDataItem iD){ set(iD.get());}
    public void set(int i) { privateValue = i;}
    public boolean isLess(GenericItemType git)// upcasts required
      { return ( privateValue < ((IntegerDataItem) git).get());}
    public boolean isEqual(GenericItemType git)
      { return ( privateValue == ((IntegerDataItem) git).get());}
    public boolean isGreater(GenericItemType git)
     { return ( privateValue > ((IntegerDataItem) git).get());}
    public int get() { return privateValue;}
    public String toString() {return ""+privateValue;}

    private int privateValue;
}
```