Marco Martinez
CISP401
Take Home Final

Question 1 - 20 pts:
The design for a generic sort was proposed
in class (see D2L week 13). Extend the design to implement the
following:
1) sort via the Quicksort() and one additional sort (you
choose) algorithm. No documentation is required
for this problem.

```
public class GenericContainer
{
   // INSTANCE VARIABLE DECLARATION
   private final int MAXSIZE = 30;
   private int sizeLimit,
               index,
               currentIndex;
   private GenericItemType[] collection;

   // CLASS CONSTRUCTORS
   // (+) GenericContainer()
   public GenericContainer()
   {
      this.collection = new GenericItemType[MAXSIZE];
      this.sizeLimit = MAXSIZE;
      this.currentIndex = 0;
   }

   // (+) GenericContainer(int size)
   public GenericContainer(int size)
   {
      this.currentIndex = 0;
      if (size <= MAXSIZE)
         this.sizeLimit = size;
      else
         this.sizeLimit = MAXSIZE;
   }

   // (+) GenericContainer(GenericContainer gc)
   public GenericContainer(GenericContainer gc)
   {
      this.currentIndex = this.index = 0;
      gc.Iterator_Initialize();
```

```
   while (gc.Iterator_hasNext())
   {
      this.collection[this.currentIndex] = gc.Iterator_getNext();
      this.index++;
   }
}

// CHANGE STATE SERVICES
// (+) void init()
public void init()
{
   Iterator_Initialize();
   while (Iterator_hasNext())
      this.collection[this.currentIndex] = null;
}

// (+) void add(GenericItemType git)
public void add(GenericItemType git)
{
   this.collection[this.index++] = git;
}

// (+) void remove(GenericItemType git)
public void remove(GenericItemType git)
{
   Iterator_Initialize();
   GenericItemType temp;
   while (Iterator_hasNext())
   {
      temp = Iterator_getNext();
      if (temp.isEqual(git))
      {
         this.collection[this.currentIndex-1] = this.collection[this.index-1];
         this.collection[this.index-1] = new IntegerDataItem();
         this.index--;
         return;
      }
   }
}

// (+) GenericItemType search(GenericItemType key)
public GenericItemType search(GenericItemType key)
{
   return biSearch(key,0,this.index);
}

// (-)  GenericItemType biSearch(GenericItemType key,int low,int high)
private GenericItemType biSearch(GenericItemType key,int low,int high)
{
   while(high >= low)
   {
```

```
      int middle = (low + high) / 2;
      if (collection[middle].isEqual(key))
      {
         return collection[middle];
      }
      if (collection[middle].isGreater(key))
      {
         return biSearch(key, low, middle-1);
      }
      if (collection[middle].isLess(key))
      {
         return biSearch(key, middle+1, high);
      }
   }
   return new StringDataItem();
}

// (+) void bubbleSort()
public void bubbleSort()
{
   int outer,inner;

   for (outer=0;outer < this.index;outer++)
      for(inner=0; inner < this.index-1;inner++)
      {
         if (collection[inner].isGreater(collection[inner+1]))
         {
            GenericItemType temp = collection[inner];
            collection[inner] = collection[inner+1];
            collection[inner+1] = temp;
         }
      }
}

// (+) void quicksort()
public void quicksort()
{
   if (this.index > 0) qSort(0, this.index-1);
}

// (-)  void qSort(int start, int finish)
private void qSort(int start, int finish)
{
   int i = start;
   int j = finish;
   GenericItemType pivot = this.collection[start + (finish - start) / 2];

   while (i <= j)
   {
      while (this.collection[i].isLess(pivot))
      {
```

```
            i++;
         }

         while (this.collection[j].isGreater(pivot))
         {
            j--;
         }

         if (i <= j)
         {
            GenericItemType temp = this.collection[i];
            this.collection[i] = this.collection[j];
            this.collection[j] = temp;
            i++;
            j--;
         }
      }

      if (start < j)
      {
            qSort(start, j);
      }
      if (i < finish)
      {
            qSort(i, finish);
      }
}

// (+) int getMax()
public int getMax()
{
   if (this.sizeLimit != 0)
      return sizeLimit;
   else
      return MAXSIZE;
}

// (+) int getLength()
public int getLength()
{
   return this.index;
}

// (+) int getCurrentIndex()
public int getCurrentIndex()
{
   return this.currentIndex;
}

// (+) GenericItemType get(int i)
public GenericItemType get(int i)
```

```
   {
      return this.collection[i];
   }

   // (+) void Iterator_Initialize()
   public void Iterator_Initialize()
   {
      this.currentIndex = 0;
   }

   // (+) boolean Iterator_hasNext()
   public boolean Iterator_hasNext()
   {
      return this.currentIndex <= this.index-1;
   }

   // (+) GenericItemType Iterator_getNext()
   public GenericItemType Iterator_getNext()
   {
      return this.collection[this.currentIndex++];
   }
}
```

Question 2 - 20 pts:
Define the following terms and show an example of the Java syntax required to implement each term.

Inheritance
Abstract Class
Polymorphism
Interface

Inheritance: The relationship between a parent and a child class. The child class inherits the parent's methods, while gaining the ability to customize itself with new classes if it deems it necessary. With the implementation of abstract classes, it gains a further ability to override the parent class' methods that are abstract with its own interpretation. This relationship is signified by the "extends" keyword.

```
public class Hourly extends Employee
```

Abstract Class: A class that contains one or more abstract methods. The purpose is to allow for child customization of the abstract Method(s). There is a hefty cost, though. An abstract class waives its right to constructors and thus necessitates Upcasting.

```
public abstract class GenericItemType
{
   public abstract boolean isLess(GenericItemType git);
```

```
    public abstract boolean isEqual(GenericItemType git);

    public abstract boolean isGreater(GenericItemType git);
}
```

Polymorphism: The implementation of overloads, overrides, abstract, and generics. The purpose of polymorphism is to have a class or program that can adapt "morph" to a variety of situations and data types "poly".

Overloads: Methods with the same name but have different signatures. The purpose is to have methods that can adapt to the data being provided to them.

```
public void set(int i)
public void set(double i)
```

Overrides: Methods that have the same name and the same signature. The purpose it to have child classes that can customize themselves to utilize their specialties however they see fit.

```
public class abstract Math
{
	public void abstract calc()
}

public class specificMath extends Math
{
	public void calc()
	{
		this.answer = this.a + this.b
	}
}
```

Abstracts: a means by which a class can implement overrides to customize itself. Signified by keyword "abstract".

```
public class abstract Math
{
	public void abstract calc()
}

public class specificMath extends Math
{
	public void calc()
	{
		this.answer = this.a + this.b
	}
}
```

Inheritance: A relationship described by a parent, child relationship. The child “inherits” the characteristics while gaining the ability to customize itself in whatever way it the creator sees fit. It is signified by the “extends” keyword.

```
public class abstract Math
{
	public void abstract calc()
}

public class specificMath extends Math
{
	public void calc()
	{
		this.answer = this.a + this.b
	}
}
```

Interface: The ability by which a user interacts with a class. This is typically done through one of three ways:

Constructors: These retain the same name of the class and are used to create or reject objects.

```
public class PotatoSmasher
{
	private String type;

	public PotatoSmasher()
	{
		this.type = new String(“Unknown potato.”);
	}
}
```

Manipulators: These allow the user to manipulate objects that have been created.

```
public class PotatoSmasher
{
	private String type;

	public void setType(String newType)
	{
		this.type = new String(newType);
	}
}
```

Accessors: These allow the user to receive information that the creator of the class deems appropriate.

```
public class PotatoSmasher
{
	private String type;

	public String getType()
	{
		return this.type;
	}
}
```

Question 3 - 20 pts:
The following geometric shapes have area calculations defined as follows:

Square	length^2
Rectangle	length * width
Circle	PI * r^2
Triangle	½ * base * height

Extend the Shape class design to accommodate the input, calculate and print needs of the entire “family” of shapes defined above and additional, unspecified shapes to be defined later. Your design must accommodate new shapes with minimal code rewrite.

```
Class Shape
(-)theShape type Shape
    Shape()
    Shape(p1,p2, ...pn)
    Shape(Shape)
    Input( .... )
    Calculate()
    Print()
```

Hierarchy Chart

Classes

Square
Circle
Rectangle
Triangle
Shape
GenericItemType
GenericContainer
AppDriver

Associations

Shape(1) --- inherits --- (1) GenericItemType
Triangle(1) --- inherits --- (1) Shape
Cricle(1) --- inherits --- (1) Shape
Square(1) --- inherits --- (1) Shape
Rectangle(1) --- inherits --- (1) Shape
GenericContainer(1) --- contains --- (m) GenericItemType
AppDriver(1) --- uses --- (1) GenericContainer

GenericContainer Class Attributes

CLASS CONSTRUCTORS
(+) GenericContainer()
(+) GenericContainer(int size)
(+) GenericContainer(GenericContainer gc)

CHANGE STATE SERVICES
(+) void init()
(+) void add(GenericItemType git)
(+) void remove(GenericItemType git)
(+) GenericItemType search(GenericItemType key)
(-) GenericItemType biSearch(GenericItemType key,int low,int high)
(+) void sort()
(-) void qSort(int start, int finish)
(+) void Iterator_Initialize()

READ STATE SERVICES
(+) int getMax()
(+) int getLength()
(+) int getCurrentIndex()
(+) GenericItemType get(int i)
(+) boolean Iterator_hasNext()
(+) GenericItemType Iterator_getNext()

Parent Abstract Class "GenericItemType" Attributes

(+) Abstract boolean isLess(GenericItemType)
(+) Abstract boolean isEqual(GenericItemType)
(+) Abstract boolean isGreater(GenericItemType)

Parent Abstract Class "Shape" Attributes

INSTANCE VARIABLE DELCARATIONs
(#) double area

(SUPER) CONSTRUCTORS
(+) Shape()
(+) Shape(Shape)

CHANGE STATE SERVICES
(+) abstract void calcArea()

READ STATE SERVICES
(+) double getArea()
(+) String toString()

Child "Square" Attributes

INSTANCE VARIABLE DECLARATIONS
(-) double length

CLASS CONSTRUCTORS
(+) Square()
(+) Square(int)
(+) Square(float)
(+) Square(double)
(+) Square(Square)

CHANGE STATE SERVICES
(+) void calcArea()

READ STATE SERVICES
(+) double getLength()
(+) String toString()

Child "Rectangle" Attributes

INSTANCE VARIABLE DECLARATIONS
(-) double length
(-) double width

CLASS CONSTRUCTORS
(+) Rectangle()
(+) Rectangle(int,int)
(+) Rectangle(float,float)
(+) Rectangle(double,double)
(+) Rectangle(Rectangle)

CHANGE STATE SERVICES
(+) void calcArea()

READ STATE SERVICES
(+) double getLength()
(+) double getWidth()

(+) String toString()

Child "Circle" Attributes

CONSTANT DEFINITIONS
(-) PI

INSTANCE VARIABLE DECLARATIONS
(-) double radius

CLASS CONSTRUCTORS
(+) Circle()
(+) Circle(int)
(+) Circle(float)
(+) Circle(double)
(+) Circle(Circle)

CHANGE STATE SERVICES
(+) void calcArea()

READ STATE SERVICES
(+) double getRadius()
(+) String toString()

Child "Triangle" Attributes

INSTANCE VARIABLE DECLARATIONS
(-) double base
(-) double height

CLASS CONSTRUCTORS
(+) Triangle()
(+) Triangle(int,int)
(+) Triangle(float,float)
(+) Triangle(double,double)
(+) Triangle(Circle)

CHANGE STATE SERVICES
(+) void calcArea()

READ STATE SERVICES
(+) double getLength()
(+) double getWidth()
(+) String toString()

AppDriver Attributes

(-) GenericContainer myShapes

(+) getInput(GenericContainer myShapes)
(+) printOutput(GenericContainer myShapes)

State Model

GenericContainer
GenericContainer() → s0
GenericContainer(Int) → s0
GenericContainer(GenericContainer) → s0

S0 → init() → s0
S0 → add(GenericItemType) → s0
S0 → remove(GenericItemType) → s0
S0 → search(GenericItermType) → s(Err)
S0 → sort() → s1
S0 → getMax() → s(Terminal)
S0 → getLength() → s(Terminal)
S0 → getCurrentIndex() → s(Terminal)
S0 → get(int) → s(Terminal)
S0 → Iterator_Initialize() → s0
S0 → Iterator_hasNext() → s(Terminal)
S0 → Iterator_getNext() → s(Terminal)

S1 → init() → s0
S1 → add(GenericItemType) → s0
S1 → remove(GenericItemType) → s0
S1 → search(GenericItermType) → s(Err)
S1→ sort() → s1
S1 → getMax() → s(Terminal)
S1 → getLength() → s(Terminal)
S1 → getCurrentIndex() → s(Terminal)
S1 → get(int) → s(Terminal)
S1 → Iterator_Initialize() → s0
S1 → Iterator_hasNext() → s(Terminal)
S1 → Iterator_getNext() → s(Terminal)

Square
Square() → s1
Square(int) → s1
Square(float) → s1
Square(double) → s1
Square(Square) → s1

S1 → calcArea() → s1
S1 → getLength() → s(Terminal)
S1 → toString() → s(Terminal)

Rectangle
Rectangle()
Rectangle(int,int)
Rectangle(float,float)
Rectangle(double,double)
Rectangle(Rectangle)

S1 → calcArea() → s1
S1 → getLength() → s(Terminal)
S1 → getWidth() → s(Terminal)
S1 → toString() → s(Terminal)

Circle
Circle()
Circle(int)
Circle(float)
Circle(double)
Circle(Circle)

S1 → calcArea() → s1
S1 → getRadius() → s(Terminal)
S1 → toString() → s(Terminal)

Triangle
Triangle()
Triangle(int,int)
Triangle(float,float)
Triangle(double,double)
Triangle(Triangle)

S1 → calcArea() → s1
S1 → getLength() → s(Terminal)
S1 → getWidth() → s(Terminal)
S1 → toString() → s(Terminal)