Second exam review

Note that there may be typos or mistakes. Yes. I want you to fail ☺ Have fun~

File I/O: String & files & printWriter
Make a file and scan through it

```
// input.txt is name of file
File f = new File("input.txt");
Scanner stdin = new Scanner(f);
String output = null;
while(stdin.hasNext())
{
	// input goes in a String
	String next = stdin.next();
	// if next word is "hello"
	if( next.equals("hello"))
		// replace it by "bye"
		output += "bye";
	else
		output += next + "\r\n";
}
PrintWriter out = new PrintWriter(f);
out.print(output);
```

Reverse Array:

```
public static int[] reverse(int[] list){
	int[] result = new int[list.length];
	for(int i=0, j<result.length-1; i<list.length; i++, j--){
		result[j]=list[i];
	}
	return result;
}
```

Linear search compares an array one by one
Binary search splits the whole array in half and compare mid #. This requires an array that is in order already.

```
public static void main(String[] args){
	int[][] int2d = new int[3][3];
	for(int[] row: int2d){
		for(int val: row){
			System.out.print(val);
		}
	System.out.println();
	}
}
```

```
public class Number implements Cloneable, Comparable{
        public int compareTo(object o){
                Number num = (Number) o;
                return n – num.getN();
        }
        // Equivalent to
        public int compareTo(object o){
                Number num = (Number) o;
                if(this.n < num.getN())
                        return -1;
                else if(this.n == num.getN())
                        return 0;
                else return 1;
        }
        public  Object clone(){
                try{
                        return super.clone();
                }
                catch(CloneNotSupportedException e){
                        return null;
                }
        }
        public static void main(String[] args){
                Number one = new Number();
                Number two =  (Number) one.clone();
                One.setone(4);
                System.out.print(two.getarr()[0]);
        }
}
```

instanceOf: the object extends another object[class]
Object o = new Student(); <- valid
ArrayList
public - accessible everywhere
protected – inherits and same package
default – same package
private – only that class

final static double PI = 3.14159;
Interface
        All data fields are public static final
        All methods are public abstract

Selection Sort: looks for the smallest integer and move it to the beginning position.

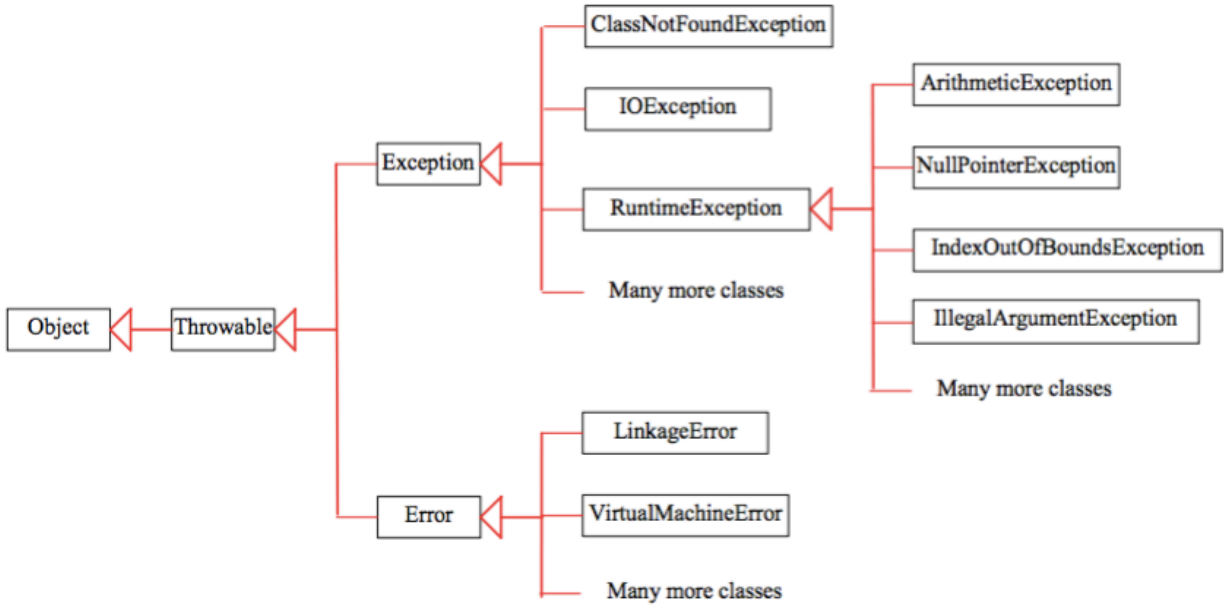
```
public static void selectionSort(double[] list){
        for(int i=0; i<list.length; i++){                                   3 2 9 1
                // find the min in the list [i….list.length]                 1 2 9 3
                double currentMin = list[i];                                1 2 9 3
                int currentMinIndex = i;                                    1 2 3 9
                for(int j=i+1; j<list.length; j++){                         1 2 3 9
                        if(currentMin > list[j]){
                                currentMin = list[j];
                                currentMinIndex = j;
                        }
                }
                // swap list[i] with list[currentMinIndex] if necessary
                if(currentMinIndex != i){
                        list[currentMinIndex] = list[i];
                        list[i] = currentMin;
                }
        }
}
```

Insertion Sort: from i to a.length, if the # is smaller than the previous number, put it in the correct position

```
public static void insertionSort(double[] a){
        int temp;
        for(int i=1; i<a.length; i++){
                for(int j=i; j>0; j--){
                        if(a[j] < a[j-1]{                                   3 2 9 1
                                temp = a[j];                                2 3 9 1
                                a[j]=a[j-1];                                2 3 9 1
                                a[j-1] = temp;                              1 2 3 9
                        }
                }
        }
}
```

Bubble Sort: i and i+1. Trade if i+1 is larger. Else, go to next

```
public static void bubbleSort(double[] a){                                  3 2 9 1
        for(int i=0; i<a.length; i++){                                      2 3 9 1
                for(int j=0; j<a.length-1; j++){                            2 3 9 1
                        if(a[j]>a[j+1]){                                    2 3 1 9
                                int temp = a[j];                            2 3 1 9
                                a[j] = a[j+1];                              2 1 3 9
                                a[j+1]=temp;                                2 1 3 9
                        }                                                   1 2 3 9
                }                                                           1 2 3 9
        }                                                                   1 2 3 9
}
```



RunTime caught when running (divide zero)
Errors are unchecked. Can't do anything but notify user an error occurred.

```
public class <ExceptionName> extends Exception{
        public<ExceptionName>(String name){
                super(name);
        }
}
```

```
public int compareTo(Object o) {
        Student test = (Student) o;
        // compare grade
        if (this.gpa > test.getGPA())
          return 1;
        if (this.gpa < test.getGPA())
          return -1;
        else
          return 0;

        // Compare Strings ALWAYS do it this way
        String a = this.name;
        String b = test.getName();
        int compare = a.compareTo(b);
        if (compare > 0)
          return 1;
        if (compare < 0)
          return -1;
        else
          return 0;
}
```

```
public class Student implements Cloneable, Comparable{
	private String name;
	private double[] grades;

	public Student(String name, double[] grades){
		this.name = name;
		this.grades = grades;
	}

	public void setName(String name){
		this.name = name;
	}
	public void setGrade(double[] grades){
		this.grades = grades;
	}
	public double[] getGrades(){
		return this.grades;
	}
	public String getName(){
		return this.name;
	}

	public double avgGrade(){
		double total = 0;
		for(int i=0; i<grades.length; i++)
			total += grades[i];
		return total / grades.length;
	}

	public static void sort(Student[] student){
		for(int i=0; i<grades.length; i++){
			for(int j=0; j<grade.length-1; j++){
				if(student[j].avgGrade()>student[j+1].avgGrade()){
					Student temp = student[j];
					student[j] = student[j+1];
					student[j+1] = temp;
				}
			}
		}
	}
	public object clone(){
		try{
			return Student(getName(), getGrades());
		}
		catch(CloneNotSupportedException ex){
			System.out.println("You suck only on the weekends. 9-4 business hours");
		}
	}
	public boolean equals(Student s){
		return (this.name.equals(s.getName()) && Array.equals(this.grades, s.getGrades()));
	}
	// Compares grades
	public int compareTo(Object o){
		Student a = (Student) o;
		Return grades - a.getGrades();
	}
}
```