1. Data structure is an efficient and logical way of storing and organizing data so that they could be used easily. An example would be an array of Integers or ArrayList of Strings.

2. Pseudo-code is a way to write programs with a mixture of natural language and programing language so that people with knowledge of any programming language will be able to understand.

3. Max in array（pseudo-code)
   Input = Integer array[n]
   Output = max
   Set Integer max = array[0]
   For i ← 1; i<array length; i ← i+1 do
       If array[i] > max then
           Set max to array[i]
       End if
   End for
   print max

4. Unstructured data includes our daily use of data such as text messages, videos, voice messages. They have yet been converted into structured and readable data for computers to read. Structured data includes arrays, ArrayList, and these data structures are structured ways of storing and organizing data. Semi-structured data are combinations of structured and unstructured data, and one perfect example is email. It includes unstructured text for the body but structured contact information.

5. Inheritance is a “is a” relationship among data types. Objects are able to inherit the state and behavior of another one. We can derive a new child class from the existing parent class, and all the methods from the parent class are inheritable for the child class.

   In the example, class Apple extends Fruits, meaning that it is accessible to all its methods. When a.colors is called, it goes into the child class and find colors method, therefore “red” is printed. Notice that when a.name is called, even though it is not in Apple, it goes to the parent class Fruits to find the method name, so it does not cause any errors.

```
package quiz1Examples;

public class Fruits {
	public void colors()
	{
		System.out.println("colors");
	}
	public void name()
	{
		System.out.println("name");
	}
```

```
}

package quiz1Examples;

public class Apple extends Fruits{
	public void colors()
	{
		System.out.println("red");
	}
}

package quiz1Examples;

public class Main {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Fruits a = new Apple();
		a.name();
		a.colors();
	}
}
```

Output will be:
name
red

6. The actual type is the class that actually creates the object, while the defined type, also called the declare type, is only used in the declaration. Define type, also known as declared type is checked during compile time whether the methods are accessible to an object of the type. The actual method called depends on the actual type at run-time.

   In the example, the declared type of b is Fruit but its actual type is Banana.

```
package quiz1Examples;

public class Fruits {
	public void colors()
	{
		System.out.println("colors");
	}
	public void name()
	{
		System.out.println("name");
	}
}
package quiz1Examples;

public class Banana extends Fruits{
	public void colors()
	{
		System.out.println("yellow");
	}
```

```
}
package quiz1Examples;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Fruits b = new Banana();
		b.colors();
	}
}
```
Output:yellow

7. Polymorphism means to have multiple forms. All java objects are polymorphism. It makes sure that methods are called for an object for a specific type when the object is disguised as a more general type. Inheritance is already a great example of polymorphism.

Polymorphism:

```
package quiz1Examples;

public class Fruits {
	public void colors()
	{
		System.out.println("colors");
	}
	public void name()
	{
		System.out.println("name");
	}
}
package quiz1Examples;

public class Apple extends Fruits{
	public void colors()
	{
		System.out.println("red");
	}

}
package quiz1Examples;

public class Banana extends Fruits{
	public void colors()
	{
		System.out.println("yellow");
	}

}

package quiz1Examples;

public class Main {
```

```
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Fruits a = new Apple();
		a.name();
		a.colors();
		Fruits b = new Banana();
		b.colors();

	}
}
```
Output:
```
name
red
yellow
```

Overload:

Method overloading is when a class has more than one method with the same name and different parameters. In the class Orange, the method weight include integer weight and double weight to be calculated. When an integer is input in the first slot, the first method with first varibale being integer will be called, and when a double is input at first variable, the second one will be called.

```
package quiz1Examples;

public class Fruits {
	public void colors()
	{
		System.out.println("colors");
	}
	public void name()
	{
		System.out.println("name");
	}
}
```
```
package quiz1Examples;

public class Orange extends Fruits{
	public void weight(int a, int b)
	{
		System.out.println("the weight of these oranges:"+ a*b);
	}
	public void weight(double a, int b)
	{
		System.out.println("the weight of these oranges:"+ a*b);
	}

}
```
```
package quiz1Examples;

public class Main {
```

```
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Orange o = new Orange();
		o.weight(2,3);
		o.weight(2.5,3);

	}
}
```

The output of the exmaple will be:
the weight of these oranges:6
the weight of these oranges:7.5

Override:
Method overriding refers to when a class has more than one method with the same name and same parameter while they are performing different tasks. Usually, if a subclass has the same method as in the parent class, the method is overridden. Therefore, inheritance is a great example of override. In the example here, Banana class inherits from Fruits, therefore the method colors is overridden.

```
package quiz1Examples;

public class Fruits {
	public void colors()
	{
		System.out.println("colors");
	}
	public void name()
	{
		System.out.println("name");
	}
}
```

```
package quiz1Examples;

public class Banana extends Fruits{
	public void colors()
	{
		System.out.println("yellow");
	}

}
```

```
package quiz1Examples;

public class Main {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		Fruits b = new Banana();
		b.colors();
	}
}
```

Output: yellow

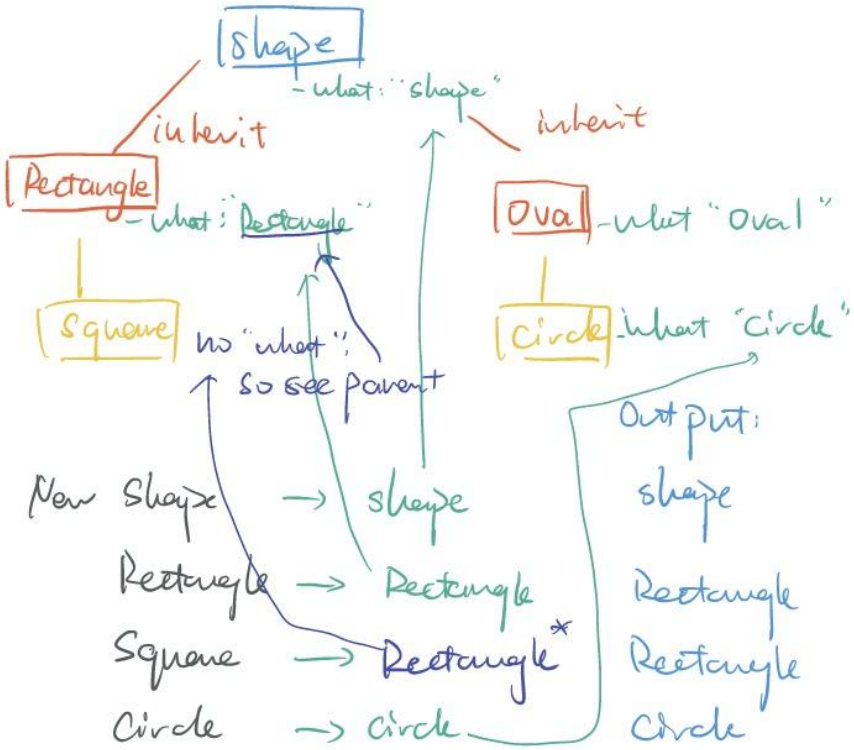
8. Trace

Integer array **a** is set to a new array of integers with a length of **10**

The method: methodOne is called, taking variable integer array a into the method
Integer array **b** is created inside the method as a new integer array with a length of **5**
The variable **a** is set to be equal to b, which is an empty integer array with a length of **5**
So in the methodOne call, **Length of a = length of b=5**
The two system.out.println prints the length of a and b, which is 5 for both
Output:55
The method methodOne ends, so the memory of array a inside and array b inside the method is gone.

Since array of integer is a primitive data type, the a in the method call does not change the array a outside of the method. The method does not do anything to this array a outside of the method. Therefore the last print statement prints the array a with a length of 10, which is the first one we created and remain unchanged, so 10 is printed.
Output:5510

9.



* for square, since there is no "what" method, its parent class Rectangle is checked, which has a "what" method and prints Rectangle
- other classes all have according "what" method. so all print class name.

10.
I have made the course class and some of the methods in course management and reports. I do not have much prior knowledge on The overall design is basically separated into two main parts, the course class and the method allowing students and admins each to perform their commands, and the User class that contains interfaces student and admin, each with access to different methods. The data of students and courses will be stored in ArrayLists with all their information. Serilization is a way of organzing object and convert objects into a byte stream, which can also be converted back. We need to use serialization with java.io.Serializable interface implemented to any superclasses.