1. ⓐ

```
import java.util.StringTokenizer;

public class printToken {
    public static void main (String[] args) {
        String str = "This is a test";
        StringTokenizer st = new StringTokenizer (str);
        System.out.println (st.countTokens());
        while (st.hasMoreTokens()) {
            System.out.println (st.nextTokens());
        }
    }
}
```

ⓑ ①

```
public class MyGenerics<T> {
    private T a;
    public void setObj (T a) {
        this.a = a;
    }
    public T getObj () {
        return a;
    }
```

⑪ public static void main (String [] args) {

MyGeneric<Interger> mygenerics =

new MyGenerics<>();

}

}

© An Inner class is a non-static nested class.

```
class Outer{
    int outer_x = 5;
    void test(){
        Inner inner = new Inner();
        inner.display();
    }
    class Inner{
        int z = 60;
        void display(){
            System.out.println(outer_x);
        }
    }
    void showz(){
        System.out.println(z);
    }
}
```

Outer class methods cannot access members of its Inner class.

(d)

(i) Here 'c' is a reference to an integer array.

'x' is just a normal variable.

(ii) Here 'c' is a reference to an integer array. (same as before)

'x' is also a reference to an integer array.

2/ @A class which contains atleast one abstract method is called an abstract class.

→ No instance can be created of an abstract class.

→ The subclass must implement the abstract class. Otherwise the subclass will be an abstract class.

```
final class X{
    final void show(){
        System.out.println("Printed from a method");
    }
}
class Y extends X{
    void show(){
        System.out.println("In subclass method");
    }
}
```

As the show() method of class X is a final method, it prevents this method from being overridden. On the other hand, class X is a final class, which prevents this class from being inherited.

(b)

```
class MyClass implements Int2 {
    public void f1() {
    }
    public void f2() {
    }
    public void f3() {
    }
}
```

(d)

Advantages of multithreading over process-based multitasking:

1. Lower overhead for communication and coordination.
2. Faster context switching.
3. More efficient use of system resources.
4. Simpler and more direct communication between threads.

3. (a) Two types of TCP sockets :-

→ Server socket : used by servers so that they can accept incoming connections from clients.

→ Socket : used by clients who wish to establish a connection to a server.

(b) Type wrappers are classes that encapsulate a primitive type within an object.

- Character
- Boolean
- Double, Float, Integer

Dynamic Method Dispatch

↳ runtime polymorphism

late binding

```
class P {
    void call() {
        System.out.println("Inside P's call method");
    }
}
class Q extends P {
    void call() {
        System.out.println("Inside Q's call method");
    }
}
class R extends Q {
    void call() {
        System.out.println("Inside R's call method");
    }
}

public class DynamicDispatchTest {
    public static void main(String[] args) {
        P p = new P(); // object of type P
        Q q = new Q(); // object of type Q
        R r = new R(); // object of type R
        P  x;          // reference of type P
        x = p;         // x refers to a P object
        x.call();      // invoke P's call
        x = q;         // x refers to a Q object
        x.call();      // invoke Q's call
        x = r;         // x refers to a R object
        x.call();      // invoke R's call
    }
}
```

For practical example please refer to **FindAreas.java**

4.@ The process of encapsulating a value within an object is called boxing.

```
Integer iOb = new Integer(100);
```

The process of extracting a value from a type wrapper is called unboxing.

```
int i= iOb.intValue();
```

(b)

	HashMap	HashTable
1. Synchronized	No	Yes
2. Thread safe	No	Yes
3. keys and values	One null keys, any null values	No permit null keys and values.

(i)

```
import java.util.HashMap;

public class Demo {
    public static void main(String[] args) {
        HashMap<String, Integer> cricketers = new HashMap<>();
        cricketers.put("Sakib Al Hasan", 300000);
        cricketers.put("Mashrafe Bin Mortoza", null);
        cricketers.put("Mushfiqur Rahim", 250000);
```

(ii)

```
        int salary = cricketers.get("Sakib Al Hasan");
        salary += 50000;
        cricketers.put("Sakib Al Hasan", salary);
    }
}
```

(c)

```
class TestClass implements serializable {
    String s;
    double d;
    public TestClass(String s, double d){
        this.s = s;
        this.d = d;
    }
}
public class TestDemo {
    public static void main(String[] args){
        TestClass obj1 = new TestClass("Mr. A", 18.5);
        //Your code
        TestClass obj2;
        //Your code
    }
}
```

```
FileOutputStream fos = new FileOutputStream("Test");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(obj1);
oos.close();
fos.close();
```

```
FileInputStream fis = new FileInputStream("Test");
ObjectInputStream ois = new ObjectInputStream(fis);
obj2 = (TestClass) ois.readObject();
fis.close();
ois.close();
```