NAME:
ID:

SCORE	Q1	Q2	Q3	Q4	TOTAL

OBJECT ORIENTED PROGRAMMING FINAL EXAM| 06.01.2020 | 75 mins

Q1 (20p): Please read the below statements carefully and put a check mark (✓) in one of the YES/NO boxes of the related question for which they hold true.

	Yes	No		Yes	No
a is a private int attribute of class ***C***. I can assign 5 to ***a*** directly in the main method from another class.	☐	☑	I can write two methods in the same class with same name and different arguments, and compiler does not give any error. This is called ***overriding***	☐	☑
If you extend an abstract class, you must implement all abstract methods.	☑	☐	A class that implements the interface must implement all the methods of the interface	☑	☐
Information: Class ***S*** is inherited from class ***P***. int ***i*** is public, int ***j*** is protected, and int ***k*** is private in ***P***. Both ***S*** and ***P*** have a method named ***f***. ***f*** of ***S*** takes an int and ***f*** of ***P*** takes a float parameter. Class ***S*** has a member of class ***R***, which has a public method ***f*** that takes no parameters. Answer the follows.					
Method ***f*** of class ***S*** can change ***i***.	☑	☐	In the main, if method ***f*** of class ***S*** is called with a float parameter, this call is directed to the ***f*** of ***P.***	☑	☐
The action in question 1.5 is called ***overloading.***	☐	☑	***i*** may be changed directly over an ***R*** object.	☐	☑
Method ***f*** of Class ***R*** overrides method ***f*** of Class ***S***	☐	☑	***j*** may be changed directly over an ***S*** object.	☑	☐

Q2(35p): In Main, we first create 4 tickets and add them all in the tickets list. Then ticketSeller takes the list and then sells them all. Finally, ticketSeller gets incomes. Sell() removes the specified ticket from the collection and compute Price. (Standard ticket price=10). To do this, write all necessary missing parts (Classes, methods, attribute) using UML and written lines of code.

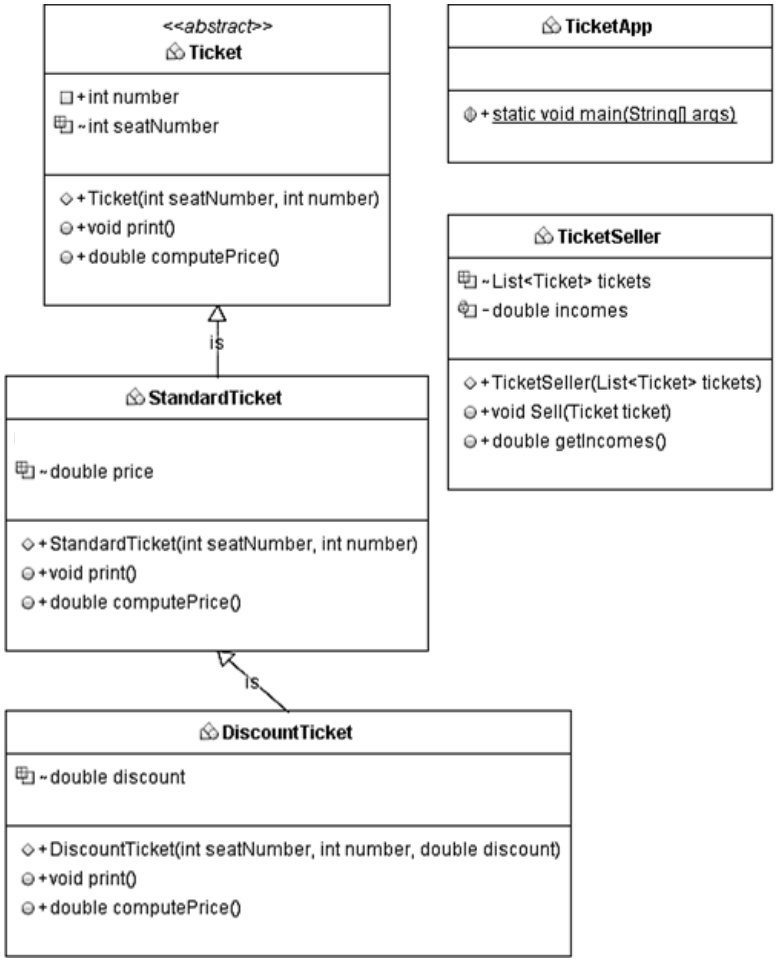
```
public class TicketApp {
public static void main(String[] args) {
Ticket t1 = new StandardTicket(10, 1);
Ticket t2 = new DiscountTicket(22, 2, 0.1);
Ticket t3 = new DiscountTicket(12, 3, 0.2);
Ticket t4 = new StandardTicket(35, 4);

List<Ticket> tickets = new ArrayList<>();
tickets.add(t1);
tickets.add(t2);
tickets.add(t3);
tickets.add(t4);


//Sorts Tickets by seat number(for Question2)
SortAndPrint(tickets);


TicketSeller ts = new TicketSeller(tickets);
ts.Sell(t1);
ts.Sell(t2);
ts.Sell(t3);
ts.Sell(t4);

System.out.println(ts.getIncomes()); } }
```



Q3 (20p): Consider the classes in Question 2, create a class named **`SeatNumberComparator`** to compare the tickets by seat number. In the class **`TicketApp`** implement the SortAndPrint method to sort the ticket collections (tickets) by seat number and print them. **(Hint: use Comparator interface) [Write your answer here!]**

```
public class SeatComparator implements Comparator<Ticket>{
   @Override
   public int compare(Ticket t1, Ticket t2) {
      return t1.seatNumber - t2.seatNumber;
   }
}

void SortAndPrint(List<Ticket> tickets){
      SeatComparator seatComp = new SeatComparator();
      Collections.sort(tickets, seatComp);
      for(Ticket t: tickets){
         t.print();
      }
}
```

Q4 (25): **Assume statements are called in main method. Fill the given table.**

```
interface A {
   void x();
   void y();
}

interface B {
   void z();
}

abstract class C implements A {
    @Override public void x(){ System.out.println(" c is doing x");}
    public abstract void y();
}
class D extends C {
    @Override public void x(){ System.out.println("d is doing x");}
    @Override public void y(){ System.out.println("d is doing y");}
    void q(){ System.out.println("d is doing q");}
}
class E extends C {
     void t(){ System.out.println("e doing t");}
     @Override public void y(){ System.out.println(" e is doing y");}
}
class F extends C implements B {
     @Override public void y(){ System.out.println(" c is doing y");}
     @Override public void z(){ System.out.println(" c is doing z");}
}
```

Statement	Compile? (yes/no)	Run? (yes/no)	If compiles or runs, explain why and write output ? If not, Correct it (if possible)
`A a1 = new E();` `a1.t();`	no	no	A1 is type of A, and A does not contain t(). So you have to downcast to E. ((E)a1).t();
`A a2 = new C();` `a2.y();`	no	no	Because C is abstract. So you cannot create an object from it.

`A a3 = new D();`
 	
`if (a3 instanceof C)`
`a3.x( );`
	Yes	Yes	Because D is A and a3 has x behaviour

`a3 = a1;` `((D)a3).q();`	yes	no	a3 is types of A from Class E. But E is not D. So wil cause run time error
`B b = new C();` `b.z();`	No	No	Because C is abstract. So you cannot create an object from it.

NAME:
ID:

Q1: Lütfen aşağıdaki ifadeleri dikkatlice okuyunuz ve ilgili sorunun YES / NO kutularından birine doğru oldukları bir onay işareti (✓) koyunuz.
Q2. Main'de önce 4 bilet olusturup ve hepsini bilet listesine ekler. ticketSeller listeyi alır ve hepsini satar ve bu satıştan gelir elde (incomes) eder. Sell () ile belirtilen biletin fiyatın hesaplanır ve listeden kaldırılır. (Standart bilet fiyatı = 10). Bu işlemleri yapmak için eksik olan kısımları (class, method, attribute) UML ve yazılı kod satırlarını yardımıyla yazın.
Q3. Soru 2'nin devamı olarak, biletleri koltuk numarasına göre karşılaştırmak için SeatNumberComparator adlı bir sınıf oluşturun. TicketApp class'ında bilet koleksiyonları (bilet) koltuk numarasına göre sıralamak ve yazdırmak için SortAndPrint methodunu doldurun. (İpucu: Comparator Interface'sini kullanın.) Cevabınızı yukarında verilen boşluğa yazınız.
Q4. Varsayım statement sütununda verilen ifadeler main'de çağrılıyor. Verilen tabloyu doldurun.

Q2.Answer here

```
public abstract class Ticket {
 public int number;
 int seatNumber;

    public Ticket(int seatNumber, int number) {
         this.seatNumber = seatNumber;
         this.number = number;}

 public abstract void print();
 public abstract double computePrice();}

public class StandardTicket extends Ticket {
    double price;
    public StandardTicket(int seatNumber, int nu
        super(seatNumber, number);
        this.price =10; }

    @Override
    public void print() {
        System.out.println(seatNumber +" "+price +" " + number);  }

    @Override
    public double computePrice() {
        return price;  }
}

public class DiscountTicket extends StandardTicket {
  double discount;
    public DiscountTicket(int seatNumber, int number, double discount) {
        super(seatNumber, number);
        this.discount = discount;}

    @Override
    public void print() {
        System.out.println(seatNumber +" "+price +" " + number + " " + discount);}

    @Override
    public double computePrice() {
        return super.computePrice() - price*discount; }
}
```

```
public class TicketSeller {
    List<Ticket> tickets;
    private double incomes;

    public TicketSeller(List<Ticket>
tickets) {
    this.tickets =tickets;}

    public void Sell(Ticket ticket){
     incomes +=ticket.computePrice();
     tickets.remove(ticket);}

    public double getIncomes(){
     return incomes;}
}
```