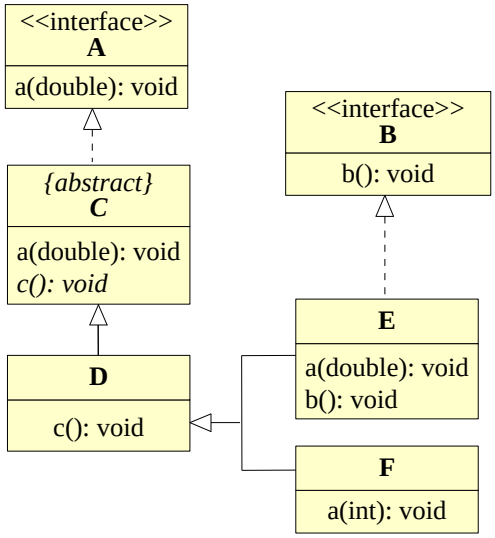
Tentamen 150819 - LÖSNINGSFÖRSLAG

Uppgift 1.

a)



b)

i) Tilldelningen A x = **new** C() ger kompileringsfel eftersom klassen C är abstrakt.

ii) Ger utskriften:

constructor i C
constructor i D
constructor i E
a() in E

iii) Ger utskriften:

constructor i C
constructor i D
constructor i F
a() in C

iv) Ger utskriften:

constructor i C
constructor i D
constructor i E
c() in D

v) Ger utskriften:

constructor i C
constructor i D
sedan inträffar ett exekveringsfel pga att den dynamiska typen på x är D och typen D kan inte typomvandlas till typen E.

vi) Ger utskriften

constructor i C
constructor i D
constructor i E
b() in E

Uppgift 2.

```
public class SingletonRandom {
    private static SingletonRandom instance = null;
    private Random random;

    private SingletonRandom() {
        random = new Random();
    }//constructor

    public static synchronized SingletonRandom getInstance() {
        if (instance == null)
            instance = new SingletonRandom();
        return instance;
    }//getInstance

    public int nextInt(int limit) {
        return random.nextInt(limit);
    }//nextInt

}//SingeltonRandom
```

Uppgift 3.

```
public class MusicPlayer {
    private Instrument instrument;
    private Volym vol = new Volym();
    public Musicplayer(Instrument instrument) {
        this.instrument = instrument;
    }
    public void lower() {
        vol.lower();
    }
    public void higher() {
        vol.higher();
    }
    public void play() {
        instrument.play();
    }
    public void changeInstrument(Instrument theInstrument) {
        instrument = theInstrument;
    }
}// MusicPlayer

public interface Instrument {
    public abstract void play();
}//Instrument

public class Trumpet implements Instrument {
    public void play() {
        //code for playing trumpet
    }
    //constructors and methoder not shown here
}//Trumpet

public class Piano implemens Instrument {
    public void play() {
        //code for playing piano
    }
    //constructors and methoder not shown here
}//Piano

public class Guitar implements Instrument {
    public void play() {
        //code for playing guitar
    }
    //constructors and methoder not shown here
}//Guitar
```

Uppgift 4.

a) Begreppet *äkta subtyp* definieras av *Liskov Substitution Principle*. Att Sub är en äkta subtyp till Sup innebär att man i ett program kan byta ut ett objekt av klassen Sup mot ett objekt av typen Sub utan att beteendet hos programmet förändras. Detta innebär att en äkta subtyp inte kan ha en svagare specifikation. Förlitar sig en klient på supertypens specifikation och subtypen har en svagare specifikation kommer det att gå galet.

b)

i) Nej! Specifikationen för metoden insertAt i klassen SortedIntList är svagare än i klassen IntList. Detta pga att förvillkoret är starkare. Förvillkoret kräver att klienten väljer ett värde på parametern i som bevarar den sorterade ordningen i listan.

ii) Nej! Specifikationen för metoden insert i klassen IntList är svagare än i klassen SortedIntList. Detta pga att eftervillkoret är svagare, då det inte finns några garantier på var det insatta element hamnar i listan.

Uppgift 5.

```
public void update(Observable o,Object arg) {
    if (o instanceof FirstClass && arg instanceof String) {
        String s = (String) arg;
        System.out.println(s.toUpperCase());
    }
    if (o instanceof SecondClass && arg instanceof Integer) {
        Integer i = (Integer) arg;
        System.out.println(i*10);
    }
}//update
```

Uppgift 6.

```
import java.util.List;
import java.util.ArrayList;
public class SortListAdapter implements Sorter {
    private NumberSorter sorter = new NumberSorter();

    @Override
    public void sort(int[] numbers) {
        List<Integer> numberList = new ArrayList<Integer>();
        for (int i = 0; i < numbers.length; i++)
            numberList.add(numbers[i]);
        sorter.sort(numberList);
        for (int i = 0; i < numbers.length; i++)
            numbers[i] = numberList.get(i);
    }
}//SortListAdapter
```

Uppgift 7.

Metoderna i ett interface är alltid publika och abstrakta, oberoende av om detta anges eller inte. Deklarationen av vårt interface ser alltså egentligen ut enligt:

```
public interface SomeInterface {
    public abstract void someMethod();
}
```

En klass som implementerar interfacet kan inte begränsa synligheten av metoderna som definieras av interfacet, vilket klassen SomeClass gör. En korrekt implementation av klassen är:

```
public class SomeClass implements SomeInterface {
    public void someMethod() {
        //some code
    }
}
```

Uppgift 8.

a) Korrekt. I en list av typen List<Number> kan ett element av typen Double läggas in eftersom Double är en suptyp till Number.

b) Kompileringsfel. List<? **extends** Number> kan antingen vara List<Number>, List<Double> eller List<Integer>. Kompilatorn vet inte vilken den specifika är och i typen List<Double> eller List<Integer> kan man inte lägga in ett objekt av typen Number.

c) Korrekt. List<? **super** Number> kan antingen vara List<Number> eller List<Object> och Integer är subtyp både till Number och Object.

d) Kompileringsfel. Typen List<? **extends** Number> inkluderar typerna List<Number> samt List<Double>, och varken Number eller Double är subtyper med typen Integer.

e) Korrekt. I alla listor av typen List<? **extends** Number> (List<Number>, List<Double> eller List<Integer>) är objekten kompatibla med typen Number.

f) Kompileringsfel. List<? **super** Number> kan antingen vara List<Number> eller List<Object>. Kompilatorn vet inte vilken den specifika är och i typen List<Object> är inte elementen kompatibla med typen Number.

g) Kompileringsfel. List<Double> är inte en subtyp till List<Number>, utan en subtyp till Collection<Double>.

h) Korrekt. List<? **extends** Number> kan antingen vara List<Number>, List<Double> eller List<Integer>, samt List<SomeType> där SomeType är en subtyp till någon av typerna Double eller Integer..

i) Korrekt. Se deluppgift h).

j) Korrek. List<? **extends** Double> kan antingen vara List<Double>, List<Number> eller List<Object>.

Uppgift 9.

```
public abstract class AirplaneDecorator implements Airplane {
    private Airplane decoratedAirplane;
    public AirplaneDecorator(Airplane decoratedAirplane) {
        this.decoratedAirplane = decoratedAirplane;
    }
    public void construct() {
        decoratedAirplane.construct();
    }
}//AirplaneDecorator

public class EjectionSeatDecorator extends AirplaneDecorator {
    public EjectionSeatDecorator(Airplane decoratedAirplane) {
        super(decoratedAirplane);
    }
    @Override
    public void construct() {
        super.construct();
        System.out.print(" Inserting Ejection Seat to airplane. ");
    }
}//EjectionSeatDecorator

/public class TurboDecorator extends AirplaneDecorator {
    public TurboDecorator(Airplane decoratedAirplane) {
        super(decoratedAirplane);
    }
    @Override
    public void construct() {
        super.construct();
        System.out.print(" Inserting Turbo to airplane. ");
    }
}//TurboDecorator
```

Alternativ lösning:

```
public abstract class AirplaneDecorator implements Airplane {
    protected Airplane decoratedAirplane;
    public AirplaneDecorator(Airplane decoratedAirplane) {
        this.decoratedAirplane = decoratedAirplane;
    }
    public abstract void construct();
}//AirplaneDecorator

public class EjectionSeatDecorator extends AirplaneDecorator {
    public EjectionSeatDecorator(Airplane decoratedAirplane) {
        super(decoratedAirplane);
    }
    @Override
    public void construct() {
        decoratedAirplane.construct();
        System.out.print(" Inserting Ejection Seat to airplane. ");
    }
}//EjectionSeatDecorator

/public class TurboDecorator extends AirplaneDecorator {
    public TurboDecorator(Airplane decoratedAirplane) {
        super(decoratedAirplane);
    }
    @Override
    public void construct() {
        decoratedAirplane.construct();
        System.out.print(" Inserting Turbo to airplane. ");
    }
}//TurboDecorator
```

Uppgift 10.

a) 6 olika utskrifer är möjliga, nämligen alla permutationer av strängarna "Thread1", "Thread2" och "Thread3", dvs

Thread1. Thread2. Thread3.
Thread1. Thread3. Thread2.
Thread2. Thread1. Thread3.
Thread2. Thread3. Thread1.
Thread3. Thread1. Thread2.
Thread3. Thread2. Thread1.

b)

i) Det kan uppstå ett s.k. *race condition*, eftersom t.ex. en tråd T1 kan anropa metoden empty() samtidigt som en annan tråd T2 anropar metoden fill(). Trådarna konkurrerar då om variabeln level. Anta att level har värdet 100.0. Följande kan då inträffa:

T1 vill tömma ur volymen 50. Eftersom level har värdet 100 går detta bra och T1 påbörjar beräkningen av satsen

```
double newLevel = level - volume;
```

Under vill T2 fylla tanken med volymen 100. T2 hinner utföra satsen

```
level = level + volume;
```

innan T1 är klar med sina beräkning. level kommer således att få värdet 150.
När sedan T1 blir klar med sin beräkning har newLevel värdet 50. T1 utför sedan satsen

```
level = newLevel;
```

och level får värdet 50. Men ett korrekt värde borde varit 100. Ett inkonsistent tillstånd har inträffat.

ii)

```
public class WaterTank {
   private double level;
   public synchronized void fill(double volume) {
      level = level + volume;
   }
   public synchronized boolean empty(double volume) {
      if (volume < level) {
         double newLevel = level - volume;
         level = newLevel;
         return true;
      }
      else
         return false;
   }
}//WaterTank
```

Uppgift 11.

a)

```
public AppStore() {
    store = new HashMap<String, Set<String>>();
}
```

b)

```
public void addApp(String osType, String appName) {
    Set<String> osTypeApps = store.get(osType);
    if (osTypeApps == null) {
        osTypeApps = new TreeSet<String>();
        store.put(osType, osTypeApps);
    }
    osTypeApps.add(appName);
}
```

c)

```
public String getOsTypeWithMaxNumApps() {
    int max = 0;
    String osTypeMaxApp = null;
    for (String osType : store.keySet()) {
        int numApps = store.get(osType).size();
        if (numApps > max) {
            max = numApps;
            osTypeMaxApp = osType;
        }
    }
    return osTypeMaxApp;
}
```

Alternativ lösning:

```
public String getOsTypeWithMaxNumApps() {
    int max = 0;
    String osTypeMaxApp = null;
    for (Map.Entry<String, Set<String>> e : store.entrySet()) {
        int numApps = e.getValue().size();
        if (numApps > max) {
            max = numApps;
            osTypeMaxApp = e.getKey();
    }
    return osTypeMaxApp;
}
```