# Instuderingsuppgifter läsvecka 2

**1.**

Vad är skillnaden mellan *implementationsarv* och *specifikationsarv*?

**2.**

Vad är *multipelt arv*?

**3.**

Arv kan användas för *specialisering* och *generalisering*. Förklara dessa begrepp.

**4.**

Varför har man i Java valt att inte stödja multipelt implementationsarv? Visa och förklara något problem som kan uppkomma vid multipelt implementationsarv.

**5.**

Förklara skillnaden mellan *abstrakta klasser* och *interface*. När är respektive mekanism att föredra?

**6.**

Vilka relationer finns definierade i UML?

**7.**

Hur definierar UML skillnaden mellan *aggregation* och *komposition*?

**8.**

Rita ett UML-diagram för att representera nedanstående klasser och interface. Försök att vara så exakt som möjligt.

```
public interface Order { . . . }
public interface Iterable { . . . }
public class StandardOrder implements Order, Iterable {
    private List<ListItem> items;
    private OrderRespository repos;
    public LineItems getItems() { . . . }
}
public class BusinessOrder extends StandardOrder { . . . }
public class OrderRespository { . . . }
```

**9.**

Vad betyder begreppet *alias*? Vad kan det uppkomma för problem när alias används?

**10.**

Hur definierar man *nya typer* i Java?

**11.**

Förklara vad en *typ* är?

**12.**

Vilka förhållanden finns mellan *supertyp* och *subtyp* avseende *v*ärden och operationer?

**13.**

Java är ett *starkt typat språk*. Vad innebär det att ett språk är starkt typat?

**14.**

Vad innebär *explicit typomvandling*? Vad finns det för faror med explicit typomvandling?

**15.**

En referensvariabel har både en *statisk typ* och en *dynamisk typ*. Förklara dessa båda begrepp!

**16.**

Förklara vad operationen **instanceof** används till?

**17.**

Förklara begreppet *polymorfism*.

**18.**

Förklara hur *statisk bindning* och *dynamisk bindning* går till.

**19.**

*The Open-Closed Principle* säger ”Software modules should be both open for extension and closed for modification”. Förklara denna princip med ett exempel!

**20.**

*The Dependency Inversion Principle* säger ”Depend upon abstractions, not upon concrete implementations”. Detta är en av de viktigaste principerna i objektorienterad design. Förklara varför.

**21.**

Förklara begreppen *overriding* (*överskuggning*), *hiding* (*döljande*) och *overloading* (*överlagring*).

**22.**

Varför bör man använda annotationen *@Override* när man överskuggar en metod?

**23.**

Vad innebär begreppet *kovarians*? Vad är fördelarna med att använda kovarians.

**24.**

Förklara begreppen *inkapsling* (*encapsulation*) och *informationsdöljande* (*information hiding*).

**25.**

Vad är en *mutator*-metod?

**26.**

Vad är en *icke-muterbar* klass?

**27.**

Är en klass icke-muterbar om klassen inte innehåller några mutator-metoder?

**28.**

Följande klasser är givna

```
public class A {
   public A() {
      System.out.println("A.A()");
   }
   public void f(A arg) {
      System.out.println("A.f(A)");
   }
   public void g(A arg) {
      System.out.println("A.g(A)");
   }
}//A
```

```
public class B extends A {
   public B() {
      System.out.println("B.B()");
   }
   public void f(B arg) {
      System.out.println("B.f(B)");
   }
   public void g(A arg) {
      System.out.println("B.g(A)");
   }
   public static void h(A arg) {      //OBS: statisk!
      System.out.println("B.h(A)");
   }
}//B
```

```
public class C extends B {
   public C() {
      System.out.println("C.C()");
   }

   public void f(A arg) {
      System.out.println("C.f(A)");
   }
   public void f(B arg) {
      System.out.println("C.f(B)");
   }
   public void g(B arg) {
      System.out.println("C.g(B)");
   }
   public static void h(A arg) {      //OBS: statisk!
      System.out.println("C.h(A)");
   }
}//C
```

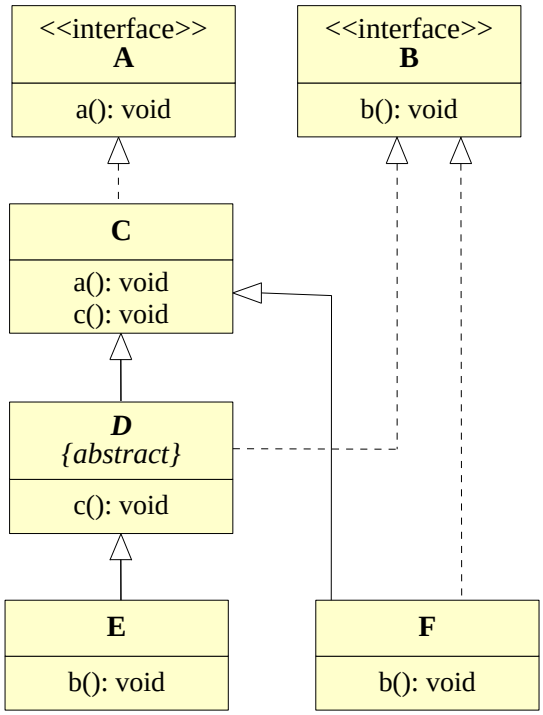
Vad blir resultatet för var och en av följande satser (ger kompileringsfel, ger exekveringsfel, skriver ut xxx, etc) ?

```
A a = new A();      (a)
A ab = new B();     (b)
B bc = new C();     (c)
C c = new C();      (d)
a.f(bc);            (e)
bc.h(c)             (f)
bc.f(ab);           (g)
c.g(a);             (h)
c.g(bc);            (i)
ab.h(ab);           (j)
ab.g(ab);           (k)
```

**29.**

Betrakta nedanstående klasser och interface:

```
public interface A {
      public void a( );
}//A

public interface B {
      public void b();
}//B

public class C implements A {
      public void a() {
            System.out.println( "a() in C" );
      }//a
      public void c() {
            System.out.println( "c() in C" );
      }//c
}//C

public abstract class D extends C implements B {
      public void c() {
            System.out.println( "c() in D" );
      }//c
}//D

public class E extends D {
      public void b() {
            System.out.println( "b() in E" );
      }//b
}//E

public class F extends C implements B {
      public void b() {
            System.out.println( "b() in F" );
      }//b
}//F
```




Vad blir resultatet för var och en av följande satser (ger kompileringsfel, ger exekveringsfel, skriver ut xxx, etc)?

**a**)
```
C x = new E();
x.c();
```

**b)**
```
Object o = new E();
System.out.println( o instanceof D );
```

**c**)
```
C x = new D();
ax.c();
```

**d**)
```
B x = new E();
D y = (D) x;
y.b();
```

**e**)
```
C x = new F();
x.b();
```

**f**)
```
B x = new F();
D y = (D) x;
x.b();
```

**g**)
```
D x = new C();
x.a();
```

**h**)
```
A x = new F();
C y = x;
y.a();
```

**30.**

Följande klasser är givna

```
public interface I {
    void f1();
}
```

```
public abstract class C1 {
    public abstract void f2();
    public void f3() { print("C1.f3"); }
    protected void print(String s) {
        System.out.println(s);
    }
}
```

```
public class C2 extends C1 implements I {
    public void f1() { print("C2.f1"); }
    public void f2() { print("C2.f2"); }
}
```

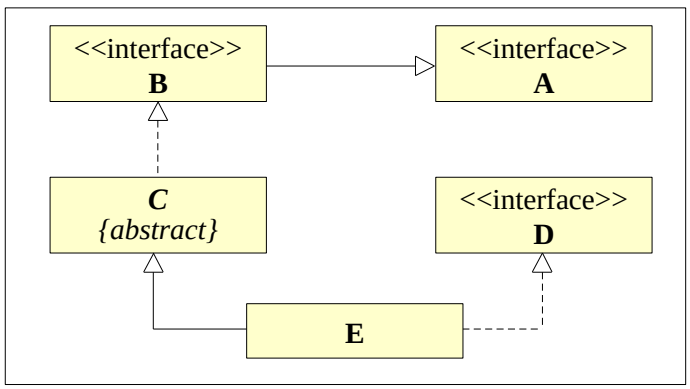
```
public class C3 extends C2 {
    public void f2() { print("C3.f2"); }
    public void f4() { print("C3.f4"); }
    public void print(String s) {
        System.out.println("++"+s+"++");
    }
}
```

```
public class Main {
    public static void func(C2 obj) {
        obj.f1();
        obj.f2();
        obj.f3();
        obj.f4();
    }
    public static void func2(C3 obj) {
        obj.f4();
    }
    public static void main(String[] arg) {
        func(new C1());
        func(new C2());
        func(new C3());
        func2(new C2());
        func2(new C3());
    }
}
```

Tre metodanrop i klassen Main ger kompileringsfel. Vilka är de, och vad är det för fel på dem? Antag nu att vi avlägsnar de felaktiga satserna och kompilerar de fem klasserna. Vad skrivs då ut om Main exekveras?

**31.**

Skriv deklarationer i Java som motsvarar nedanstående UML-diagram. Det räcker med att ange klass- respektive interfacehuvuden.



**32.**

Skriv deklarationer i Java som motsvarar nedanstående UML-diagram. Klasserna A och D skall innehålla nödvändiga instansvariabler, samt en lämplig konstruktor.