Lösningsförslag till instuderingsuppgifter läsvecka 2

1.

Den statiska typen är den typ som anges vid deklarationen av variablen. Den dynamiska typen är den typ som objekt till vilken variabeln refererar.

2.

Ordet polymorfism betyder *mångformighet*. I objektorientering innebär polymorfism att en superklass är utbytbar mot sina subklasser.

3.

Vid statisk bindning bestäms vilken metod som skall anropas vid kompileringen. Kompilatorn kontrollera att metoden finns genom att börja söka efter metoden uppåt i arvshierakin med start i den klass som anges av den statiska typen. Den första metoden med rätt signatur som påträffas väljs. Finns ingen sådan metod inträffar ett kompileringsfel.

Vid dynamisk bindning bestäms vilken metod som skall anropas under exekveringen. Kompilatorn kontrollerar att metoden finns genom att börja söka efter metoden uppåt i arvshierakin med start i den klass som anges av den statiska typen. Finns ingen sådan metod inträffar ett kompileringsfel. Vid exekveringen börjar sökningen efter metoden i klassen som specificeras av den dynamiska typen. Sökningen sker uppåt i arvshierakin och den första metoden med rätt signatur som påträffas väljs.

4.

Innebörden av *The Dependency Inversion Principle* (DIP) är att man skall programmera mot gränssnitt, inte mot konkreta klasser. Fundamentet för principen är således polymorfism. Följs DIP minskar beroendet av en specifik klass - koden blir mycket mer flexibel och återanvändbar än vad som annars skulle vara fallet.

5.

Innebörden av *The Open-Closed Principle* är att det skall vara möjligt att lägga till ny funktionalitet i ett program utan att modifiera den existerande kod. Detta realiseras oftast genom att använda *The Dependency Inversion Principle*, dvs med hjälp av polymorfism. *The Open-Closed Principle* är målet och *The Dependency Inversion Principle* är verktyget.

6.

Overriding (överskuggning) innebär att en instansmetod omdefinieras i en subklass. *Hiding* (döljande) innebär att en klassmetod omdefinieras i en subklass. *Overloading* (överlagring) innebär att en klass har flera metoder med samma namn (men med olika signaturer).

7.

Om en metod är annoterad med @Override, men metoden inte överskuggar någon metod ger kompilatorn ett kompileringsfel.

8.

Kovarians innebär att en subklass kan ersätta returtypen i en metod som överskuggas med en subtyp till returtypen som superklassen anges för den överskuggade metoden.

9.

Begreppet inkapsling (*encapsulation*) betyder att data och de operationer som kan utföras på denna data sammanförs till en enhet, dvs i det objektorienterade paradigmet till en klass. Informationsdöljning (*information hiding*) innebär extern åtkomst till datan i den inkapslade enheten (=klassen) endast kan ske via ett väldefinierat publikt gränssnitt.

10.

En mutator-metod är en metod som förändrar tillståndet hos ett objekt.

11.

Instanser av en icke-muterbar klass förändrar inte sina tillstånd under sin livstid.

12.

Nej, inte nödvändigtvis! Exempelvis kan en instansvariabel vara publikt tillgänglig, varför tillståndet i ett objekt kan förändras.

13.

a) A.A()

b) A.A()
B.B()

c) A.A()
B.B()
C.C()

d) A.A()
B.B()
C.C()

e) A.f(A)

f) B.h(a)
statisk bindning!

g) C.f(A)

h) B.g(A)

i) C.g(B)

j) Kompileringsfel
klassen A saknar metod h

k) B.g(A)

14.

a) Ger utskriften ”c( ) in D”

b) Ger utskriften ”true”

c) Tilldelningen C x = **new** D() ger kompileringsfel, eftersom klassen D är abstrakt.

d) Ger utskriften ”b( ) in E”

e) Anropet x.b() ger kompileringsfel, eftersom typen C inte har operationen b().

f) Ger exekveringsfel. Typen F kan inte typomvandlas till typen D.

g) Tilldelningen D x = **new** C() ger kompileringsfel, eftersom typen C inte är subtyp till D.

h) Tilldelningen C y = x ger kompileringsfel, eftersom x är av typen A och A är ingen subtyp till C.

15.

Fel i Main.java

rad 6: obj.f4() är fel, eftersom C2 inte har någon metod f4().

rad 12: func(**new** C1()) är fel, eftersom C1 är en abstrakt klass.

rad 15: func2(**new** C2()) är fel, eftersom C2 inte är en subtyp till C3.

Om dessa fel avlägsnas blir utskriften:

```
C2.f1
C2.f2
C1.f3
++C2.f1++
++C3.f2++
++C1.f3++
++C3.f4++
```

16.

```
public interface A {. . .}
public interface B extends A {. . .}
public interface D {. . .}
public abstract class C implements B {. . .}
public class E extends C implements D {. . .}
```

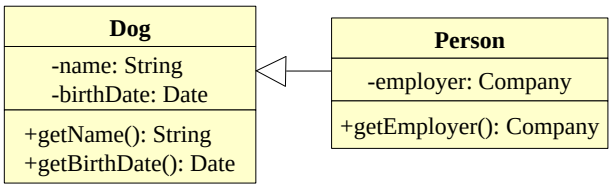
17.

```
public class A {
  private D objD = new D();
  private C objC;
  public A(C objC) {
    this.objC = objC;
  }
  . . .
}//A
public class C {
  . . .
}//C
```

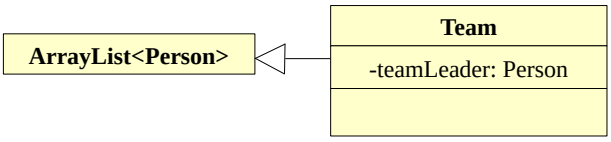
```
public class B {
  private C objC;
  public B(C objC) {
    this.objC = objC;
  }
  . . .
}//B
public class D {
  . . .
}//D
```

18.

Implementationsarv definierar is-relationen och finns inte någon sådan relation mellan sub- och superklassen skall givtevis inte implementationsarv användas.



Felaktig användning av implementationsarv, en person är ingen hund.



Felaktig användning av implementationsarv, ett team är ingen ArrayList..

19.

En subklass ärver koden från sina superklasser. Detta betyder att om någon metod överskuggas i subklassen måste kan implementationen vara beroende av hur den överskuggade metoden är implementerad I superklassen. Förändringar av implementationen i superklassen kan allså påverka tillhörande subklasser och dessa måste också förändra implementationen för att bli korrekta.

20.

The Principle of Least Astonishment säger att om en klient tror att den har en referens till ett objekt av typen Super, men i verkligheten har en referens till ett objekt av subtypen Sub, skall det inte bli några överraskningar när klienten sänder ett meddelande till objektet. Klienten skall få vad denne förväntar sig och inget annat.

21.

Liskov Substitution Principle säger att det är acceptabelt att göra klassen S till en subklass av klassen T, om och endast om det för varje publik metod som finns både i T och S gäller att S's metod som indata godtar alla värden som T's metod godtar samt att S gör alla bearbetningar på denna indata som T gör.

Javakompilatorn kan dock inte avgöra vad en metod gör. När kompilatorn avgör om en metod är korrekt överskuggad är det endast det mest basala som kontrolleras, t.ex. att den metoden inte begränsar synligheten, har rätt returvärde eller inte kastar ett nytt kontrollerade exception.

22.

Delegering innebär att ett objekt tar hjälp av ett annat objekt för att lösa sin arbetsuppgift.

23.

a) Delegering innebär en svagare koppling än implementationsarv. Finns inget beroende av koden, endast av de tjänster som tillhandahålls. Om källkoden inte finns att tillgå är delegering det enda alternativet.

b) Implementationsarv skall användas när det finns en ”is”-relation och ett behov av polymorfism.