# Övning vecka 2.

Denna vecka ska vi titta på skillnader mellan primitiva typer och referenstyper, typomvandling mellan primitiva typer samt referenser kontra kopior av referenser samt "aliasing". Vi ska som hastigast också identifiera problem i samt bygga om dåligt designad programkod.

## Uppgift 1 Anropssemantik & alias

Vad blir utskriften när main-metoden i klassen Vector nedan exekveras?

```
public class Vector {
   private int x;
   private int y;
   private int z;

   public Vector(int x, int y, int z) {
      this.x = x;
      this.y = y;
      this.z = z;
   }

   public void add(Vector v) {
      x += v.x;
      y += v.y;
      z += v.z;
   }

   public void silly(int x, int y, int z) {
      this.x = ++x;
      this.y = y + 1;
      this.z += z;
   }

   public int getX() { return x; }

   public int getY() { return y; }

   public int getZ() { return z; }

   public String toString() {
      return "Vector, <x = " + x + ", y = " + y + ", z = " + z + ">";
   }

   public static void main(String[ ] args) {
      Vector a = new Vector(1, 0, 0);
      Vector b = new Vector(0, 1, 0);
      Vector c = a;
      int x = 1;
      int y = 2;
      int z = 3;
      a.add(b);
      b.add(b);
      c.add(c);
      c.silly(x, y, z);
      System.out.println("a: " + a);
      System.out.println("b: " + b);
      System.out.println("c: " + c);
      System.out.println("x: " + x + "\ty: " + y + "\tz: " + z);
   }
}//Vector
```

**Uppgift 2 Referens eller kopia?**

Betrakta klasserna nedan. Avgör vad utskriften blir då main-metoden i klassen RefvalMain exekveras.

```
public class ValueHolder {
   public Integer value;
   public ValueHolder(Integer value) {
      this.value = value;
   }
}//ValueHolder

public class ValueHolderSwapper {
   public void swap(ValueHolder vh1,
ValueHolder vh2) {
      Integer tmp = vh1.value;
      vh1.value = v2h.value;
      vh2.value = tmp;
   }
}//ValueHolderSwapper
```

```
public class SimpleSwapper {
   public void swap(Integer x, Integer y) {
      Integer temp = x;
      x = y;
      y = temp;
   }
}//SimpleSwapper

public class ValueHolderSwapper2 {
   public void swap(ValueHolder v1,
ValueHolder v2) {
      v1 = new ValueHolder(v2.value);
      v2 = new ValueHolder(v1.value);
   }
}//ValueHolderSwapper2
```

```
public class RefvalMain {
   public static void main(String[ ] args) {
      SimpleSwapper ss = new SimpleSwapper();
      ValueHolderSwapper vhs = new ValueHolderSwapper();
      ValueHolderSwapper2 vhs2 = new ValueHolderSwapper2();
      int a = 1;
      int b = 2;
      ss.swap(a, b);
      System.out.println("a= " + a + " b= " + b);
      Integer c = new Integer(1);
      Integer d = new Integer(2);
      ss.swap(c, d);
      System.out.println("c= " + c + " d= " + d);
      ValueHolder v1 = new ValueHolder(a);
      ValueHolder v2 = new ValueHolder(b);
      vhs.swap(v1, v2);
      System.out.println("a= " + v1.value + " b= " + v2.value);
      vhs2.swap(v1, v2);
      System.out.println("a= " + v1.value + " b= " + v2.value);
   }
}//RefvalMain
```

**Uppgift 3 Implicit typomvandling**

Vid körning av main-metoden i klassen Casting nedan, skulle man kunna tro att utskriften blir 30587000000000.

I själva verket blir det något helt annat. Exakt vad är inte så intressant men genom att härleda vilken typ varje deluttryck har kan man första varför. Tips: Vilken precision har (de primitiva) typerna i Java? Vilka värden kan de representera?

```
public class Casting {
   public static void main(String[ ] args) {
      final long PARSEC = 30587 * 1000000000 * 1000;
      final long M_PER_KM = 1000;
      System.out.println(PARSEC / M_PER_KM);
   }
}//Casting
```

**Uppgift 4 Snygg design?**

Kalle Klåpare har byggt ett spel där man ska döda varelser. Strukturen på koden blev dock mindre lyckad, eftersom det kommer att bli svårt att nya lägga till varelser i spelet eller ändra beteendet hos någon varelse. Betrakta klasserna Gang och Creature nedan:

```
import java.util.*;
public class Gang {
   private List<Creature> members;
   public Gang(int size) { members = new ArrayList<Creature>(size); }
   public void add(Creature m) { members.add(m); }
   public boolean remove(Creature m) {return members.remove(m); }
   public int damageSum() {
      int total = 0;
      for (Creature c : members) {
         if (c != null) {
            int energy = c.getEnergy();
            switch (c.getType()) {
               case SNAKE: total += 10 * energy; break;
               case GOBLIN: total += 4 * energy * energy; break;
               case SPIDER: if (energy > 5) total += 100; break;
            }
         }
      }
      return total;
   }
   public static void main(String[ ] args) {
      Gang gang = new Gang(3);
      gang.add(new Creature("Sour Serpent", Creature.Type.SNAKE));
      gang.add(new Creature("Greasy Goblin", Creature.Type.GOBLIN));
      gang.add(new Creature("Spicy Spider", Creature.Type.SPIDER));
      System.out.println("Collective damage: " + gang.damageSum());
   }
}//Gang
```

```
public class Creature {
   public enum Type { SNAKE, GOBLIN, SPIDER };
   private Type type;
   private int energy = 100;
   private String name;
   public Creature(String n, Type t) {
      this.name = n;
      this.type = t;
   }
   public Type getType() { return type; }
   public int getEnergy() { return energy; }
   public void setEnergy(int e) { this.energy = e; }
   public String getName() { return name; }
}//Creature
```

a) Diskutera vilka nackdelar som finns med strukturen i klasserna Gang och Creature.

b) Skriv om koden så att man blir av med de problem ni identifierat. Kan abstrakta klasser eller interface vara till någon hjälp? I så fall, vilket borde man välja? Vill eller kan man kombinera dem? Har polymorfism något med saken att göra?

**Uppgift 5 Samma kontra lika**

Betrakta nedanstående klass:

```
public class References {
    public static void main(String[ ] args) {
        stringRefs();
        integerRefs();
    }

    private static void print(String exp, boolean b) {
        System.out.println(exp + " -> " + b);
    }

    private static void printIdEq(Object a, Object b, String aName, String bName) {
        System.out.println(aName + " == " + bName + " -> " + (a == b) + "\t"
                           + aName + ".equals(" + bName + ") -> " + a.equals(b));
    }

    private static void stringRefs() {
        String s3 = new String("False");
        String s4 = new String("False");
        printIdEq(s3, s4, "s3", "s4");
        String s5 = "True";
        String s6 = "Tr" + "ue";
        printIdEq(s5, s6, "s5", "s6");
        String s7 = "False";
        String sx = "F";
        String s8 = sx + "alse";
        printIdEq(s7, s8, "s7", "s8");
    }

    private static void integerRefs() {
        Integer i = new Integer(2);
        Integer j = new Integer(2);
        print("i >= j", i >= j);
        print("i == j", i == j);
        print("i == 2", i == 2);
    }
}//References
```

När main-metoden exekveras erhålls följande utskrifter:

```
s3 == s4 -> false       s3.equals(s4) -> true
s5 == s6 -> true        s5.equals(s6) -> true
s7 == s8 -> false       s7.equals(s8) -> true
i >= j -> true
i == j -> false
i == 2 -> true
```

Troligen är inte detta vad man förväntar sig! För att förstå varför utskriften blir som den blir måste man söka förklaringen i hur Java har implementerat klassen String och omslagsklassen Integer. Slå upp detta i *The Java Language Specification, Java SE 7 Edition* (delkapitel 3.10.5 respektive 5.1.1 – 51.8) och förklara sedan resultatet i ovanstående utskrift.