# Övning vecka 2.

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

## 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++;
        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);
    }
}
```

**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;
    }
}

public class RefvalMain {
    public static void simpleSwap(Integer x, Integer y) {
        Integer temp = x;
        x = y;
        y = temp;
    }
    public static void valueHolderSwap(ValueHolder vh1, ValueHolder vh2) {
        Integer tmp = vh1.value;
        vh1.value = vh2.value;
        vh2.value = tmp;
    }
    public static void valueHolderSwap2 (ValueHolder v1, ValueHolder v2) {
        v1 = new ValueHolder(v2.value);
        v2 = new ValueHolder(v1.value);
    }
    public static void main(String[ ] args) {
        int a = 1;
        int b = 2;
        simpleSwap(a, b);
        System.out.println("a= " + a + " b= " + b);

        Integer c = new Integer(1);
        Integer d = new Integer(2);
        simpleSwap(c, d);
        System.out.println("c= " + c + " d= " + d);

        ValueHolder v1 = new ValueHolder(1);
        ValueHolder v2 = new ValueHolder(2);

        valueHolderSwap (v1, v2);
        System.out.println("a= " + v1.value + " b= " + v2.value);

        valueHolderSwap2 (v1, v2);
        System.out.println("a= " + v1.value + " b= " + v2.value);
    }
}
```

**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 förstå 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);
    }
}
```

**Uppgift 4 Polymorfism och dynamisk typ**

a) Betrakta klasserna Binding1, Base, SubA och SubB nedan:

```
public class Binding1 {
    public static void printValue(SubA v) {
        System.out.println("This was a SubA: " + v.toString());
    }
    public static void printValue(SubB v) {
        System.out.println("This was a SubB: " + v.toString());
    }
    public static void printValue(Base v) {
        System.out.print("That was a Base");
        if (v.getClass() != Base.class) {
            System.out.print(", subclassed by " + v.getClass().getName() + ": ");
        } else {
            System.out.print(", not a strict subclass of Base." + ": ");
        }
        System.out.println(v.toString());
    }
    public static void main(String[ ] args) {
        Base spa = new Base();
        SubA apa = new SubA();
        SubB bepa = new SubB();
        Base apalt = apa;
        SubA bepalt = bepa;
        printValue(spa);
        printValue(apa);
        printValue(bepa);
        printValue(apalt);
        printValue(bepalt);
    }
}

public class Base {
    public String toString() {
        return "I am a Base!";
    }
}

public class SubA extends Base {
    public String toString() {
        return "I am a SubA!";
    }
}

public class SubB extends SubA {
    public String toString() {
        return "I am a SubB!";
    }
}
```

Klassen Binding1 innehåller metoder för utskrift av objekt av typerna Base, SubA och SubB.

Alla klasser i Java är subklasser till klassen Object som innehåller metoden toString(), och klasserna Base, SubA och SubB *överskuggar* denna metod. I klassen Binding1 är metoden printValue *överlagrad* och de olika versionerna skiljer sig åt med avseende på typen hos den formella parametern.

Vad kommer utskriften att bli när vi kör Binding1.main? Vilka metoder i vilka klasser kommer att exekveras vid körningen av Binding1.main och framförallt; varför anropas just dessa metoder?

Tips: Metoden getClass(), som returnerar ett objekt av klassen Class, ärvs från klassen Object. Klassen Class har en metod getName() som returnerar en String med namnet på klassen.

b) Betrakta klassen Binding2 nedan:

```
public class Binding2 {
   public static void main(String[ ] args) {
      Base[ ] objs1 = new Base[ ] { new SubA(), new SubB(), new Base() };
      for (Base o : objs1) {
         Binding1.printValue(o);
      }
      SubA[ ] objs2 = new SubA[ ] { new SubA(), new SubB() };
      for (Base o : objs2) {
         Binding1.printValue(o);
      }
      for (SubA o : objs2) {
         Binding1.printValue(o);
      }
   }
}
```

Antag att klasserna SubA, SubB och Base är de samma som definierades i deluppgift a). Vad blir utskriften när Binding2.main körs? Varför?

**Uppgift 5 Snygg design?**

Kalle har byggt ett spel där man ska döda varelser. Strukturen på koden (klasserna Gang och Creature nedan) blev dock mindre lyckad:

```
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 class Main {
   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());
   }
}
```

*forts.*

```
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;
   }
}
```

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? Vilket borde man i så fall välja? Har polymorfism något med saken att göra?