Making Decision

Pattern Matching

```
void compare(Number number) {
   if(number instanceof Integer data) {
      System.out.print(data.compareTo(5));
   }
}
```

```
void firstIfFalse(Object object) {
    if(!(object instanceof String s)){
      //here then s is NOT a String
      return;
    }
    //then here it is!. Mind the return first if
    System.out.println("length of string: "+s.length());
}
```

Pattern Matching - interface

The compiler can't definitively state that t is not a List. Some future program could create a subclass of `Throwable` that does implement `List`.

```
void notCompatible(Throwable t) {
  //type not compatible - t cannot be a string
  /*
  if(t instanceof String s) {}
  */
  //strange behaviour for interface!
  if(t instanceof List list) {
    System.out.println("that's weird");
  }
}
```

Pattern Matching

Pattern Matching - subtype

Pattern variable type must be a **strict subtype** of the Type under test.

```
Number number = 5;
//this is a simple instanceof, not pattern matching
if(number instanceof Number){}  //this is OK

//Pattern type 'Number' is the same as expression type
//pattern variable type must be a strict subtype of Number
//if(number instanceof Number n2){} //DOES NOT COMPILE!
//this is ok as Long is a strict subtype Of Number
if(number instanceof Long nL) {}  //OK
```

Switch

Switch Statement

The value of a case statement must be one of these:

- a constant;
- a literal value;
- a final variable (not just effectively final)

```
int aValue = 5;
final int aConstant = 6;
switch (x){
    case 2:  //ok - literal
    case aConstant:  //ok - constant
    case aValue: //does not compile, not final
}
```

WARNING: You always need the `case`, a part in the `default` case.

Multi cases

I use comma (`,`)to separate values.

```
void workOrChillOut(DayOfWeek day) {
  //with the new switch, I can have multiple cases in the same li
  switch (day) {
    case MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY:
      System.out.println("You have to work!");
      break;
    default:
      System.out.println("Enjoy your weekend!");
  }
}
```

Multi cases - default

`case` and `default` are separated by colon.

```
int x = 7;
switch (x){
    case 2, 3:
      System.out.println("2,3");
    case 7: default: case 8, 9:
      System.out.println("default");
}
```

strange switch

Switch supported types

- int,byte, short, char
- Integer, Byte, Short, Character
- enum
- String
- var (if type was previously resolved)

Switch expression

```
void printItalianDayWithDefault(DayOfWeek day) {
  var result = switch (day) {
    case MONDAY -> "lunedi";
    case TUESDAY -> "martedi";
    case WEDNESDAY -> "mercoledi";
    case THURSDAY -> "giovedi";
    case FRIDAY -> "venerdi";
    default -> "weekend";
  };
}
```

We can have a switch expression even without return value assigned.

```
int cnt = 0;
char c = 'a';
//there is no return value assigned so the default is not neede
switch (c) {
  case 'a' -> cnt++;
  case 'b' -> cnt++;
}
System.out.println(cnt); //1
```

yield

If the switch expression returns a value, then every branch that isn't an expression must yield a value.

```
final int score1 = 8, score2 = 3;
char myScore = 7;
var goal = switch (myScore) {
  default -> {
    if (10 > score1) {
      yield "unknown";
    } else {
    {yield "ciao";}  //brackets not needed
    }
}
```

Switch expression with yield