



# Java traineeship

Strings en Arrays



# Leerdoelen

- Het aanmaken en manipuleren van String en StringBuilder objecten
- Veelgebruikte methodes van de String en StringBuilder klasse leren gebruiken
- Één- en multidimensionale arrays aanmaken en gebruiken
- Declareren, aanmaken en gebruik van ArrayList en het begrip van de voordelen van een ArrayList tov Arrays
- Methodes leren gebruiken waarmee je elementen kan toevoegen, verwijderen en aanpassen in een ArrayList
- Omgaan met tijdobjecten in Java

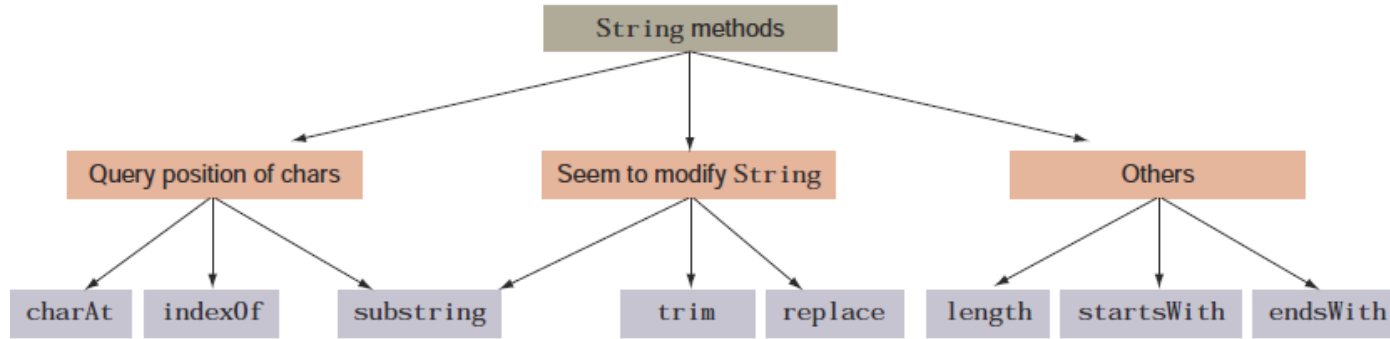
# Strings

We hebben natuurlijk in de afgelopen lessen al gewerkt met Strings, maar vandaag zullen we ons helemaal verdiepen in de wereld van Strings.

Allereerst: Strings zijn gedefinieerd in de Java API. Om specifieker te zijn; in de `java.lang` package.

Verder is een String niets anders dan een rij van characters achter elkaar. Het eerste character staat op index 0 (waarom?) en de index van de laatste character is dan natuurlijk de lengte van de String - 1

# Methoden van de String klasse



# charAt()

```
String naam = new String("Paul");  
System.out.println(naam.charAt(0)); // P  
System.out.println(naam.charAt(2)); // ?  
System.out.println(naam.charAt(10)); // ?
```

# indexOf()

```
String letters = "ABCAB";  
System.out.println(letters.indexOf('B'));    // 1  
System.out.println(letters.indexOf("S"));    //  
-1  
System.out.println(letters.indexOf("CA"));    // ?  
  
System.out.println(letters.indexOf('B', 2)); // ?
```

# substring()

```
String examen = "Oracle";  
String sub = examen.substring(2);  
System.out.println(sub); // acle  
  
String resultaat = examen.substring(2,  
4);  
System.out.println(resultaat); // ?
```

# trim()

```
String varMetSpaties = " AB CB ";  
System.out.print(":");  
System.out.print(varMetSpaties);  
System.out.print(":"); // : AB CB :  
  
System.out.print(":");  
System.out.print(varMetSpaties.trim());  
System.out.print(":"); // :AB CB:
```



# replace()

```
String letters = "ABCAB";  
System.out.println(letters.replace('B', 'b')); // ABCAb  
System.out.println(letters.replace("CA", "12")); // ?
```

# length()

```
System.out.println("bob".length()); //  
3  
  
System.out.println("").length();    //  
?
```

# startsWith() en endsWith()

```
String letters = "ABCAB";  
System.out.println(letters.startsWith("AB"));    // true  
System.out.println(letters.startsWith("a"));      // false  
System.out.println(letters.startsWith("A", 3));   // ?  
  
System.out.println(letters.endsWith("CAB"));     // ?  
System.out.println(letters.endsWith("B"));       // ?  
System.out.println(letters.endsWith("b"));       // ?
```

# equals()

Bij het vergelijken van twee niet-primitieve datatypen kun je gebruikmaken van de `.equals()` methode.

```
String tekst = "Hallo Wereld Hallo Wereld";  
String tekst2 = "Hallo Wereld";  
if(tekst.equals(tekst2 + " " + tekst2)) {  
    System.out.println("Gelijk!");  
} else {  
    System.out.println("Ongelijk!");  
}  
// Output: ?
```

# String opdrachtje

## Opdracht

Gegeven een willekeurige string als invoer, reverse deze string:

Abcdef -> fedcba

*Bonus:* Geef ook aan wanneer een string een palindroom is.

Lepel -> lepel

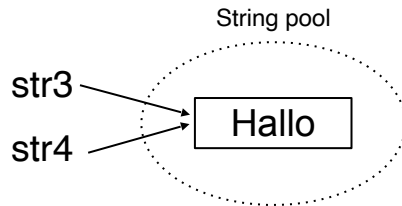
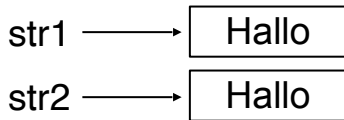
Palindroom!

# Het aanmaken van Strings

Laten we eerst eens kijken naar hoe Strings aangemaakt worden. Dit kan zoals je misschien al verwachtte op verschillende manieren:

```
String str1 = "Hallo";  
String str2 = "Hallo";  
String str3 = new String("Hallo");  
String str4 = new String("Hallo");  
System.out.println(str1 == str2); // true  
System.out.println(str3 == str4); // false
```

Hoe kan dit?



# Het aanmaken van Strings

Ook kunnen we Strings aanmaken met de `StringBuilder` en `StringBuffer` klassen:

```
StringBuilder sb1 = new StringBuilder("String Builder");  
String str5 = new String(sb1); // De constructor van String accepteert een StringBuilder.  
StringBuffer sb2 = new StringBuffer("String Buffer");  
String str6 = new String(sb2); // De constructor van String accepteert een StringBuffer.
```

# Testje: Tel het aantal String Objecten

```
class ContString {  
    public static void main(String... args) {  
        String zomer = new String("zomer");           // 1  
        String zomer2 = "zomer";                       // 2  
        System.out.println("zomer");                   // 3  
        System.out.println("herfst");                   // 4  
        System.out.println("herfst" == "zomer");        // 5  
        String herfst = new String("zomer");           // 6  
    }  
}
```



# De klasse String is immutable

Het concept dat de klasse String onveranderlijk is, is een belangrijk punt om te onthouden. Eenmaal gemaakt, kan de inhoud van een object van de klasse String nooit worden gewijzigd.

De onveranderlijkheid van String-objecten helpt de JVM String-objecten opnieuw te gebruiken, waardoor de geheugenoverhead wordt verminderd en de prestaties worden verbeterd.

# Mutable strings: StringBuilder

De klasse `StringBuilder` is gedefinieerd in het pakket `java.lang` en heeft een **mutable** reeks aan characters.

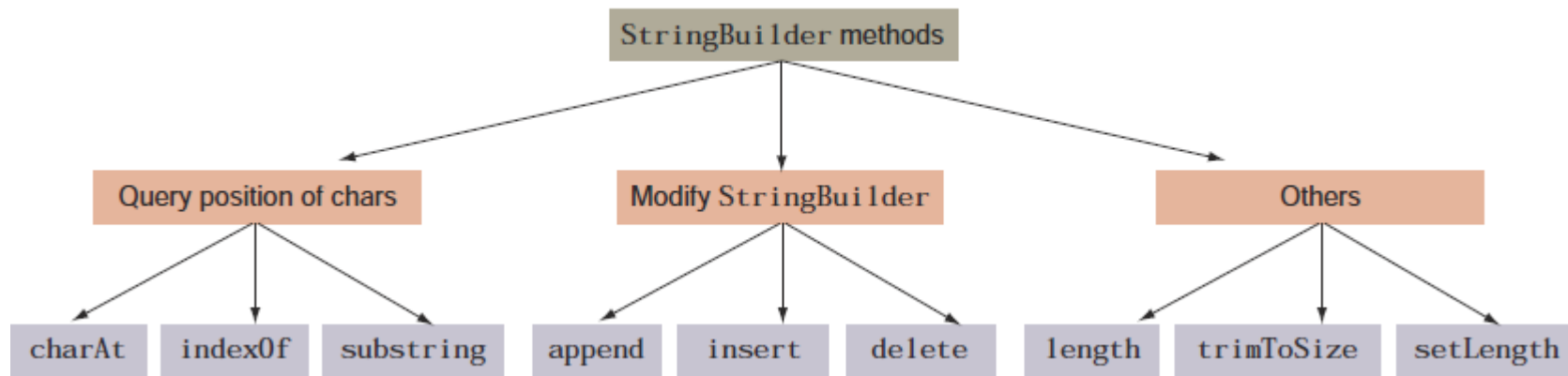
Je moet de klasse `StringBuilder` gebruiken als je te maken hebt met grotere tekenreeksen of als je de inhoud van een tekenreeks vaak wijzigt. Als je dit doet, verbeter je de prestaties van je code.

# Het aanmaken van StringBuilder objects

Je kan als volgt objecten van de klasse StringBuilder maken met behulp van meerdere overloaded constructors:

```
class MaakStringBuilderObjects {  
    public static void main(String args[]) {  
        StringBuilder sb1 = new StringBuilder();  
        StringBuilder sb2 = new StringBuilder(sb1);  
        StringBuilder sb3 = new StringBuilder(50);  
        StringBuilder sb4 = new StringBuilder("Shreya  
Gupta");  
    }  
}
```

# Methoden van de StringBuilder klasse



We zijn nu met name geïnteresseerd in de methoden die gebruikt worden om Strings aan te passen

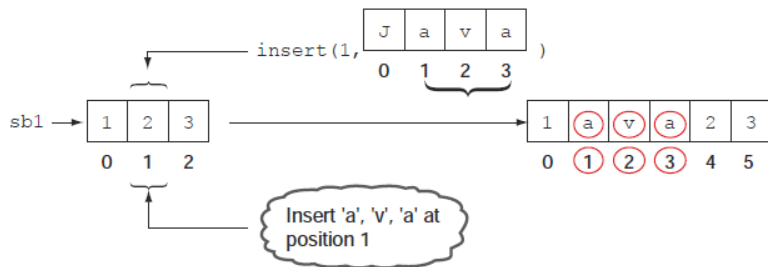
# Append()

```
class AppendStringBuilder {  
    public static void main(String args[]) {  
        StringBuilder sb1 = new StringBuilder();  
        sb1.append(true);  
        sb1.append(10);  
        sb1.append('a');  
        sb1.append(20.99);  
        sb1.append("Hi");  
        System.out.println(sb1); // true10a20.99Hi  
    }  
}
```

# insert()

```
class InsertStringBuilder {  
    public static void main(String args[]) {  
        StringBuilder sb1 = new StringBuilder("Bon");  
        sb1.insert(2, 'r');  
        System.out.println(sb1); // ?  
    }  
}
```

```
StringBuilder sb1 = new StringBuilder("123");  
char[] naam = {'J', 'a', 'v', 'a'};  
sb1.insert(1, naam, 1, 3);  
System.out.println(sb1); // ?
```



# delete() en deleteCharAt()

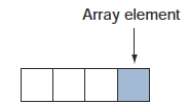
```
class DeleteStringBuilder {  
    public static void main(String args[]) {  
        StringBuilder sb1 = new StringBuilder("0123456");  
        sb1.delete(2, 4); // delete vanaf index 2 tot 4 (inclusief)  
        System.out.println(sb1); // ?  
    }  
}  
  
// of verwijder alleen een enkele char  
class DeleteStringBuilder {  
    public static void main(String args[]) {  
        StringBuilder sb1 = new StringBuilder("0123456");  
        sb1.deleteCharAt(2);  
        System.out.println(sb1); // ?  
    }  
}
```

# Wat is een Array?

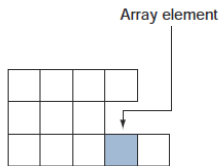
Een array is een object dat een verzameling waarden opslaat. Het feit dat een array zelf een object is, wordt vaak over het hoofd gezien. Ik herhaal: **een array is zelf een object**; het slaat verwijzingen op naar de gegevens die het opslaat.

Arrays kunnen twee soorten gegevens opslaan:

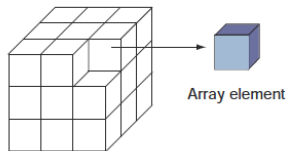
- Een collectie van primitieve data types
- Een collectie van objecten



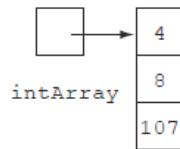
One-dimensional array



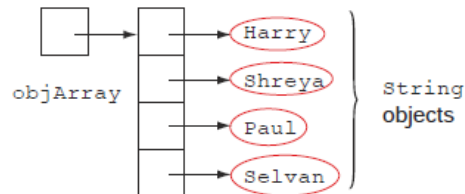
Two-dimensional array



Three-dimensional array



Array of primitive data



Array of objects



# Arrays

Met behulp van arrays kunnen we meerdere geheugenlocaties voor een datatype in één keer alloceren. Zo kunnen we bijvoorbeeld 10 integers alloceren:

```
int[] getallen = new int[10];
```

Deze 10 integers hebben een locatie in de array: de index. De index loopt van 0 tot de grootte van de array (0 t/m 9 in dit voorbeeld). Met behulp van blokhaken kunnen we een element aanspreken:

```
getallen[4] = 2  
System.out.println(getallen[4]) // 2
```

Getallen[4] is dus het **vijfde** element van de array

# Arrays (2)

Eigenschappen van arrays in Java:

- 'Pure' array: index van opeenvolgende integers
- Begint met index 0.
- Homogeen datatype (wat is dat?)
- Benaderen van een element is efficiënt  $O(1)$  (wat is dat?)
- Geïnitieerd bij allocatie op een 'nulwaarde'
  - Array van booleans: false
  - Array van shorts/bytes/integers/longs: 0
  - Array van doubles/floats: 0.0
  - Array van objecten: null

# Declareren en later initialiseren van het array

Je kunt ook eerst alleen een array declareren en later initialiseren. Dan moet je wel het `new` keyword gebruiken.

```
int[] getallen;  
  
// .. andere, zeer belangrijke code  
  
getallen = new int[] {10,20,30,40,50,60,71,80,90,91};  
System.out.println(getallen[2] + getallen[8]); // 30 + 90 = 120
```

# De vier manieren van een array alloceren

1. Declareren en direct alloceren, initialisatie met de standaardwaarde:

```
int[] getallen = new int[10];
```

2. Declareren en direct alloceren, zelf initialiseren:

```
int[] getallen = {10,20,30,40,50,60,71,80,90,91};
```

3. Declareren en later alloceren, initialisatie met de standaardwaarde:

```
int[] getallen;  
// ..  
getallen = new int[10];
```

4. Declareren en later alloceren, zelf initialiseren:

```
int[] getallen;  
// ..  
getallen = new int[]  
{10,20,30,40,50,60,71,80,90,91};
```

# Lengte en sommatie

Het aantal elementen in de array kunnen we opvragen met behulp van `.length`  
Met bijvoorbeeld behulp van een for-loop kunnen we nu over een array  
'lopen' (meer over for-loops later)

```
int[] getallen = {10,20,30,40,50,60,71,80,90,91};  
int som = 0;  
for(int i = 0; i < getallen.length; i++) {  
    som = som + getallen[i];  
}  
System.out.println(getallen.length); // 10  
System.out.println(som);              // 542
```

# Arrays

## Opdracht:

Schrijf een programma met de naam `PrintArray` dat de gebruiker vraagt om het aantal items in een array (een niet-negatief geheel getal) en het opslaat in een int-variabele met de naam `NUM_ITEMS`. Vervolgens wordt de gebruiker gevraagd om de waarden van alle items en worden deze opgeslagen in een int-array met de naam `items`. Het programma zal dan de inhoud van de array afdrukken in de vorm van `[x1, x2, ..., xn]`

```
Voer het aantal items in: 5
Voer de waarde van alle items in (gescheiden door spatie): 3 2 5 6 9
De waarden zijn: [3, 2, 5, 6, 9]
```

### Bonus

```
***(3)
**(2)
...
```

## Bonus:

Laat het programma de inhoud van de array in een grafische vorm afdrukken, waarbij de array-index en waarden worden weergegeven door het aantal sterren

# Array van objecten

We hadden het er net over gehad dat we naast primitieven ook objecten in een array kunnen stoppen:

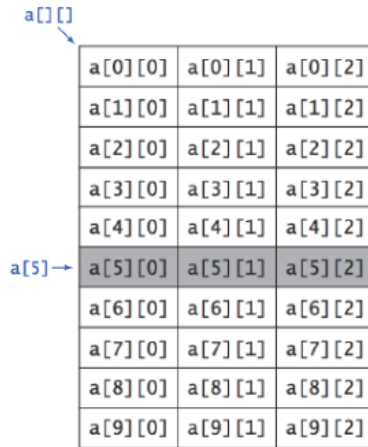
```
Breuk[] breuken = {new Breuk(1,2), new Breuk(3,5), new  
Breuk(7,14)};  
for(Breuk breuk : breuken)  
    System.out.println(breuk); // 1/2, 3/5, 7/14
```

Arrays van objecten moeten alleen wel geïnitialiseerd worden.

```
Breuk[] breuken = new Breuk[3];  
for(Breuk breuk : breuken)  
    System.out.println(breuk);
```

# Two-dimensionale arrays

```
int M = 10;
int N = 3;
double[][] a = new double[M][N];
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        a[i][j] = 0.0;
    }
}
```



The diagram shows a 10x3 array 'a' represented as a table. The rows are indexed from 0 to 9, and the columns from 0 to 2. A blue arrow points to the first column header 'a[i][0]', and another blue arrow points to the row index '5' in the first column. The row containing 'a[5][0]', 'a[5][1]', and 'a[5][2]' is shaded gray.

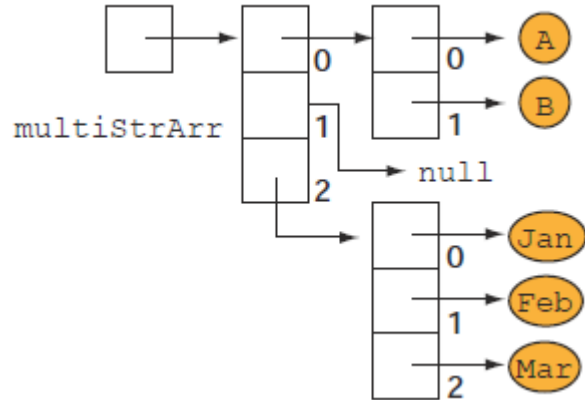
a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]
a[3][0]	a[3][1]	a[3][2]
a[4][0]	a[4][1]	a[4][2]
a[5][0]	a[5][1]	a[5][2]
a[6][0]	a[6][1]	a[6][2]
a[7][0]	a[7][1]	a[7][2]
a[8][0]	a[8][1]	a[8][2]
a[9][0]	a[9][1]	a[9][2]

*A 10-by-3 array*



# Asymmetrische multidimensionale arrays

```
String multiStrArr[][] = new String[][] {  
    {"A", "B"},  
    null,  
    {"Jan", "Feb",  
    "Mar"}},  
};
```



# ArrayList

Waarom zou ik me druk maken om een ArrayList als ik al objecten van hetzelfde type in een array kan opslaan?

Nou zoals je nu hopelijk weet zijn arrays statisch van lengte. Je kan niet de array groter of kleiner maken dan de lengte waarmee je dat hebt geïnitieerd.

Dit is waar ArrayLists in het beeld komen. Dit zijn namelijk schaalbare arrays.

# ArrayList

Een **arrayList** is dus een lijst van objecten welke dynamisch kan worden uitgebreid.

Omdat een ArrayList een ingebouwd type is, maar we toch een lijst van willekeurige objecten moet kunnen opslaan (generiek type), moeten we hinten welk type we willen opslaan.

We gaan later verder in op generieke types, maar daar hebben we eerst nog wat meer voorkennis voor nodig.

# ArrayList initialiseren

Op onderstaande manier kun je een ArrayList voor Strings aanmaken:

```
import java.util.ArrayList;
// verderop
ArrayList<String> lijst = new ArrayList<String>();
```

Dit geeft een lege lijst waar we vervolgens objecten van het type **String** in kunnen stoppen.

# Objecten in een ArrayList stoppen

Vervolgens kunnen we met de **add** methode van de ArrayList een object van het aangegeven type aan de ArrayList toevoegen:

```
ArrayList<String> lijst = new ArrayList<String>();  
lijst.add("Hallo");  
lijst.add("Wereld");  
  
lijst.add(1, "WELKOM"); // wat voor effect heeft  
dit?
```

# Objecten weer uit een ArrayList halen

Een ArrayList werkt ook in de foreach loop (later meer) net zoals een array:

```
ArrayList<String> lijst = new ArrayList<String>();  
lijst.add("Hallo");  
lijst.add("Wereld");  
for(String woord : lijst)  
    System.out.println(woord);
```

# Objecten weer uit een ArrayList halen (2)

Alle Strings krijgen een plekje in de ArrayList en met behulp van de **get**-methode kun je een String op de gegeven positie er weer uit halen.

```
ArrayList<String> lijst = new ArrayList<String>();  
lijst.add("Hallo");  
lijst.add("Wereld");  
for(int i = 0; i < lijst.size(); i++)  
    System.out.println(lijst.get(i));
```

# Aanpassen van elementen in een ArrayList

Alle Strings krijgen een plekje in de ArrayList en met behulp van de **get**-methode kun je een String op de gegeven positie er weer uit halen.

```
ArrayList<String> lijst = new ArrayList<String>();  
lijst.add("Hallo");  
lijst.add("Wereld");  
lijst.set(1, "Wereld!!!!"); // vervangt de waarde op index 1
```



# Positie in de ArrayList bepalen

Met behulp van **indexOf** kun je kijken of een element al in de ArrayList zit, en zo ja wat de eerste positie is waar hij zich bevindt. Hiervoor zal Java de **equals** methode van het object gebruiken.

```
ArrayList<String> lijst = new ArrayList<String>();  
lijst.add("Hallo");  
lijst.add("Wereld");  
System.out.println(lijst.indexOf("Wereld")); // 1  
System.out.println(lijst.indexOf("Hello")); // ?
```

# Elementen uit een ArrayList verwijderen

Verder kun je met `remove(int index)` een element op een gegeven positie uit de ArrayList halen en met `remove(Object o)` de eerste instantie van dat Object uit de ArrayList.

**Het aanpassen van een ArrayList (dingen toevoegen, aanpassen of verwijderen) terwijl je er doorheen itereert, is een slechte programmeergewoonte en levert onverwacht gedrag op!**

# Objecten vergelijken voor gelijkheid

Aan het begin van deze slides, zagen we hoe de klasse `String` een set regels definieerde om te bepalen of twee `String`-waarden gelijk zijn en hoe deze regels werden gecodeerd in de methode `equals`

Naast `Strings` kunnen we ook twee objecten met elkaar vergelijken en controleren of deze gelijk aan elkaar zijn. Dit kunnen we doen met dezelfde `equals` methode:

# Objecten vergelijken voor gelijkheid

De `equals` methode is gedefinieerd in de classe `java.lang.Object`. Alle Java-klassen erven direct of indirect deze klasse. De implementatie van de `equals` methode ziet er daarbinnen als volgt uit:

```
public boolean equals(Object obj) {  
    return (this == obj);  
}
```

Deze code vergelijkt alleen of twee objectvariabelen naar hetzelfde object verwijzen. Hoe krijgen we het dan toch voor elkaar om gelijkheid van onze objecten te vergelijken?

# Objecten vergelijken voor gelijkheid

```
Punt punt1 = new Punt(1, 1);  
Punt punt2 = new Punt(1, 1);  
System.out.println(punt1 == punt1); // true  
System.out.println(punt1 == punt2); // false
```

We zien dat punt1 en punt2 niet naar hetzelfde object verwijzen en daarom de '==' false teruggeeft. Om er voor te zorgen dat we hier toch true terug krijgen, zullen we onze eigen equals methode moeten definiëren in de Punt klasse. Deze overschrijft dan de standaard methode:

```
class Punt {  
    boolean equals(Punt punt) {  
        return x == punt.x && punt.y == y;  
    }  
}  
  
System.out.println(punt1.equals(punt2)); // true
```

# Werken met agendagegevens

Dan even heel wat anders... Als programmeur zal je vroeg of laat eens te maken krijgen met data en tijd objecten. Daarom besteden we in het laatste deel van deze les hier nog kort wat aandacht aan.

# LocalDate

Om datums op te slaan, zoals een verjaardag of jubileum, een bezoek aan een plaats of het starten van een baan, school of universiteit, hoef je geen tijd op te slaan. `LocalDate` zal in dit geval perfect werken. `LocalDate` kan worden gebruikt om datums zoals 2015-12-27 zonder tijd of tijdzones op te slaan.

```
LocalDate datum1 = LocalDate.of(2015, 12, 27);  
LocalDate datum2 = LocalDate.of(2015, Month.DECEMBER, 27);
```

We kunnen ook de huidige datum ophalen met:

```
LocalDate datum3 = LocalDate.now();
```

Ook kunnen we een datum aanmaken en die verder gebruiken:

```
LocalDate datum4 = LocalDate.parse("2025-08-09");
```

# LocalDate.getXXX()

We kunnen instantiemethoden zoals getXX() gebruiken om dingen op te vragen zoals het jaar, de maand, etc.

```
LocalDate datum = LocalDate.parse("2020-08-30");  
System.out.println(datum.getDayOfMonth()); // 30  
System.out.println(datum.getDayOfWeek()); // SUNDAY  
System.out.println(datum.getDayOfYear()); // 243  
System.out.println(datum.getMonth()); // AUGUST  
System.out.println(datum.getMonthValue()); // 8  
System.out.println(datum.getYear()); // 2020
```



# LocalDate.isAfter() en LocalDate.isBefore()

We kunnen ook data met elkaar vergelijken:

```
LocalDate verjaardagBob = LocalDate.parse("2002-08-30");  
LocalDate verjaardagJan = LocalDate.parse("2002-07-29");  
System.out.println(verjaardagBob.isAfter(verjaardagJan)); // true  
System.out.println(verjaardagBob.isBefore(verjaardagJan)); // false  
System.out.println(verjaardagBob.isBefore(verjaardagBob)); // false
```

# Datum veranderen

We kunnen ook een huidige datum aanpassen door er dagen, weken of maanden bij op te tellen of af te trekken:

```
LocalDate verjaardag = LocalDate.of(2052,03,10);  
System.out.println(verjaardag.minusDays(10)); // 2052-02-29  
System.out.println(verjaardag.minusMonths(2)); // 2052-01-10  
System.out.println(verjaardag.minusWeeks(30)); // 2051-08-13  
System.out.println(verjaardag.minusYears(1)); // 2051-03-10  
  
System.out.println(verjaardag.plusDays(1)); // 2052-03-11  
System.out.println(verjaardag.plusMonths(1)); // 2052-04-10  
System.out.println(verjaardag.plusWeeks(7)); // ?  
System.out.println(verjaardag.plusYears(1)); // 2053-03-10
```

# LocalTime

Naast data kunnen we ook tijd representeren in een object. Dit doen we in het `LocalTime` object. Dit kan op meerdere niveaus:

```
LocalTime timeHrsMin = LocalTime.of(12, 12);  
LocalTime timeHrsMinSec = LocalTime.of(0, 12, 6);  
LocalTime timeHrsMinSecNano = LocalTime.of(14, 7, 10,  
998654578);
```

Ook kunnen we net als `LocalDate` ook vragen om de huidige tijd en zelf een tijd instellen:

```
LocalDate date3 = LocalTime.now();  
LocalTime time = LocalTime.parse("15:08:23");
```

# LocalTime constants

LocalTime kent een aantal waarden die altijd hetzelfde zijn. Deze komen zo nu en dan van pas:

```
LocalTime.MIN           // Minimale tijd in deze tijdzone (00:00:00)
LocalTime.MAX           // Maximale tijd in deze tijdzone (23:59:59.999999999)
LocalTime.MIDNIGHT      // Waar de nacht begint (00:00:00)
LocalTime.NOON          // Waar de middag begint (12:00:00)
```

# LocalTime getXXX()

Opnieuw kunnen we ook verschillende elementen van de LocalTime ophalen zoals:

```
LocalTime tijd = LocalTime.of(16, 20, 12, 98547);  
System.out.println(tijd.getHour());    // 16  
System.out.println(tijd.getMinute());  // 20  
System.out.println(tijd.getSecond());  // 12  
System.out.println(tijd.getNano());    // 98547
```

# Tijd veranderen

Net zoals de dagen kunnen we ook tijd manipuleren met de methodes

`minusHours()`, `minusMinutes()`, `minusSeconds()` en `minusNanos()`

`plusHours()`, `plusMinutes()`, `plusSeconds()`, en `plusNanos()`

# LocalDateTime

We kunnen ook datum en tijd combineren. Dan gebruiken we `LocalDateTime`. Dit object kan van alle voorgenoemde methodes gebruik maken.

# Eindopracht: galgje

1. Lees het woord van de gebruiker.
2. Maak een `ArrayList<Character>` zo lang als de lengte van het woord, met alleen onderstrepingsstekens '\_'. Dit vertegenwoordigt de letters die momenteel door de gebruiker worden geraden. Wanneer een gebruiker een letter in een woord raadt, wordt de schatting bijgewerkt en worden letters weergegeven die met succes zijn geraden.
3. Zolang het woord nog niet geraden is:
  - a. vraag de gebruiker om een teken.
  - b. update de huidige schatting met de geraden letters in woord.
  - c. druk de nieuwe schatting af.
4. Zodra iemand het woord heeft geraden of het aantal beurten voorbij zijn stopt het spel.



# Leerdoelen

- ✓ Het aanmaken en manipuleren van String en StringBuilder objecten
- ✓ Veelgebruikte methodes van de String en StringBuilder klasse leren gebruiken
- ✓ Één- en multidimensionale arrays aanmaken en gebruiken
- ✓ Declareren, aanmaken en gebruik van ArrayList en het begrip van de voordelen van een ArrayList tov Arrays
- ✓ Methodes leren gebruiken waarmee je elementen kan toevoegen, verwijderen en aanpassen in een ArrayList
- ✓ Omgaan met tijdobjecten in Java

# Vragen?

- E-mail mij op [joris.vanbreugel@code-cafe.nl](mailto:joris.vanbreugel@code-cafe.nl)!
- Join de Code-Café community op discord!

