Java Collections Framework

- Ett ramverk för att hantera datastrukturer – samlingar av objekt
- Detta ramverk ingår i Javas standardbibliotek sedan Java 2 (version 1.2)
- Består av interfaces, abstrakta klasser och konkreta klasser för att hantera och manipulera datastrukturer
- Är uppdelade i 3st huvudkategorier: **Set**, **List** och **Map**
- Alla dessa finns i *java.util* -paketet

Collection

- Ett interface som definierar en container (behållare) för att hålla och manipulera grupper av objekt (dessa kallas ofta för elements)
- Är super-interface till **List** och **Set**
- Tillhandahåller grundläggande metoddefinitioner för att lägga till eller ta bort element
- Klasser som implementerar Collection kan kasta ett *UnsupportedOperationException* om metoden inte kan implementeras av klassen
- Collections växer dynamiskt

Set

- Ett interface som utökar Collection
- Beskriver en Collection som **inte innehåller några dubbletter av element** dvs. två element, e1 och e2 får inte vara: e1.equals(e2) == true
- Kan som mest innehålla ett null-element - vissa implementationer förbjuder dock detta
- Definierar inga nya metoder utöver de som ärvs från Collection
- Om mutable-objekt sparas måste man tänkta på att det då kan förekomma dubbletter
- Tre konkreta implementationer av Set är: **HashSet**, **LinkedHashSet** och **TreeSet**

HashSet

- Konkret klass som implementerar Set
- Är både **osorterad** och **oordnad** - ordningen av elementen är inte garanterad att vara konstant
- Objekt som lagras i en HashSet måste implementera **hashCode()** på ett effektivt sätt
- Ger bra prestanda om hashCode() implementeras rätt på elementen

LinkedHashSet

- Konkret klass som implementerar Set och är subklass till HashSet
- Är **osorterad** men **ordnad** - ordningen av elementen baseras på insättningsordning
- Denna implementation används om iterations-ordningen är viktig

TreeSet

- Konkret klass som implementerar Set
- Är **sorterad** efter "natural order" eller dina egna "comparsion rules"
- Eftersom den är sorterad är den **också ordnad** (sortering är en typ av ordering)
- Element som läggs till måste implementera **Comparable**-interface - detta ger "natural ordering"

  eller

- När ett TreeSet skapas skickas ett objekt av en klass som implementerar **Comparator** in som argument

List

- Ett interface som utökar Collection
- Beskriver en Collection som innehåller en **ordnad samling** av element
- Definierar metoder för att manipulera en Collection baserat på index-värden
- Denna **kan innehålla dubbletter** - detta är dock inget krav
- Kan innehålla flera element som är null - vissa implementationer förbjuder dock detta
- Varje element får ett index när det läggs till
- Ett element kan läggas in på och hämtas från ett visst index (index börjar på 0)
- Två konkreta implementationer av List är: **ArrayList** och **LinkedList**

ArrayList

- Konkret klass som implementerar List
- Är som en växande array - använder internt en vanlig array som lagring
- Är **osorterad** men **ordnad** - ordering sker efter index-värdet som varje element får
- Används när snabb iteration är av värde
- Använd inte om insättning och borttagning ofta görs av element som inte ligger sist

LinkedList

- Konkret klass som implementerar List
- Är **osorterad** men **ordnad** - ordering sker efter index-värdet som varje element får
- Används om snabb insättning och borttagning av element var som helst i listan är av värde
- Tillhandahåller metoder för att lägga till och ta bort element från början eller slutet av listan

Iterator och ListIterator

- Interface som ger möjlighet att stega igenom element i en Set eller List
- **ListIterator** ger möjlighet att stega igenom en **List** både framåt och bakåt
- **Iterator** ger möjlighet att ta bort element från den **Collection** den representerar
- **ListIterator** ger också möjlighet att lägga till element från den **List** som den representerar

Map

- Kopplar nycklar till värden
- Nycklarna är som indexvärden för en List
- En Map kan inte innehålla dubbletter av nycklar
- Map interfacet definierar metoder för förfrågningar, uppdateringar tillgång till nycklar och värden

Map fortsättnig

- Ett nyckelvärdepar kallas för ett **Entry**
- Map tillhandahåller metoder för att få tre olika Collectionvyer: *nyckel*, *värde* och *Entry*.
- Map har ett subinterface **SortedMap** som sorterar nycklarna i en Map
- Map tillhandahåller inte stöd för Iterators
- Iterators får du genom dina collection-vyer
- Tre konkreta implementationer av Map är: **HashMap**, **LinkedHashMap** och **TreeMap**

LinkedHashMap

- Konkret klass som implementerar Map
- Ordnad efter insättningsordning
- Access order

HashMap

- Konkret klass som implementerar Map
- Är inte ordnad
- Bra prestanda vid operationer på att hitta, sätta in och ta bort en *entry*

TreeMap

- Konkret klass som implementerar Map och SortedMap
- Bra för att stega igenom en collections nyckelvy
- Sorterarad efter nycklarna genom *Comparable* samt *Comparator*

Collections

- Static Factory-klass
- Hjälpklass för Collections-ramverket
- Mest stöd för List
- Har metoder för att manipulera Collection-klasser – **reverse, shuffle, copy**
- Kan skapa Singelton colletions – (Singelton - en instans per JVM)
- Kan skapa read-only colletions
- Kan skapa synchronized collections

Arrays

- Hjälpklass för vanliga arrayer
- Kan t.ex. konvertera en array till en List
- Kan sortera och kopiera arrayer

Metoden hashCode()

- Finns definierad i java.lang.Object klassen – alla ärver denna metod
- Om du definierar equals-metoden i din klass ska du alltid definiera din egen hashCode-metod
- Tre regler vid skapande av en bra hashCode-metod

    1. Ditt objekt måste ge samma hash vid upprepad exekvering (om du inte ändrar på ditt objekt)
    2. Om två objekt är lika enligt dess equals-metod måste hashCode() ge lika hash-resultat
    3. Två objekt som ger samma resultat från hashCode behöver inte var lika enligt equals()

Metoden hashCode() fortsättning

- Exempel på hashCode():

```
// Datamedlemmar hos en klass
private int userId = 1001;
private int age = 900;
private String username = ”Yoda”;
...
public int hashCode()
{
        int result = 1;                    // ta ett tal, inte noll
        result = 37 * result + userId ; // multiplicera med ett primtal plus din variabel
        result = 37 * result + age;        // gör likadant med dina andra variabler
        result = 37 * result + username.hashCode();
        return result;
}
```

- Om du har objekt som attribut i ditt objekt anropa hashCode() metoden på dem och lägg detta till resultatet