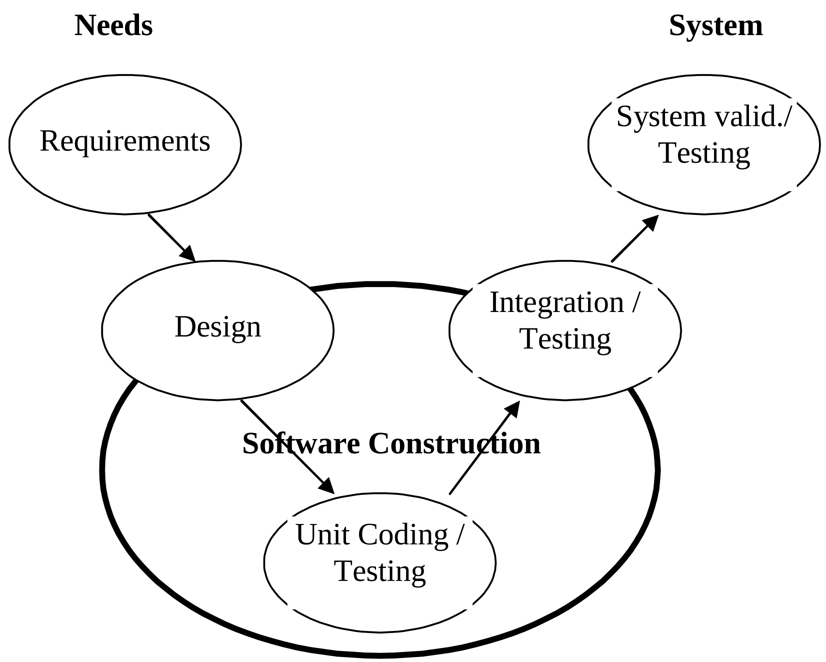
FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 1 – SUPPORT PRESENTATION (PART 1)

Outline

- Software development process
- Java Collections Framework
- Polynomial Theory
- Additional resources – polynomial arithmetic

Software Development Process



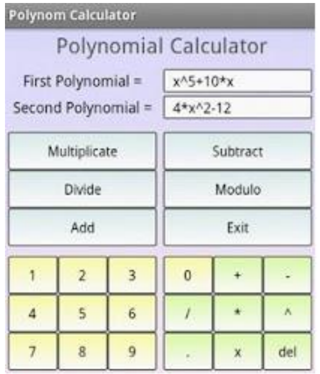
Problem and solution

PROBLEM: “Performing polynomial operations on paper is difficult and time consuming.”

$$\begin{array}{r} 2x^2 + 3x + 1 \\ +\quad x^3 + 6x^2 - 5 \\ \hline x^3 + (2x^2 + 6x^2) + 3x + 1 - 5 \end{array}$$

How to design and implement the solution?

SOLUTION: Polynomial calculator

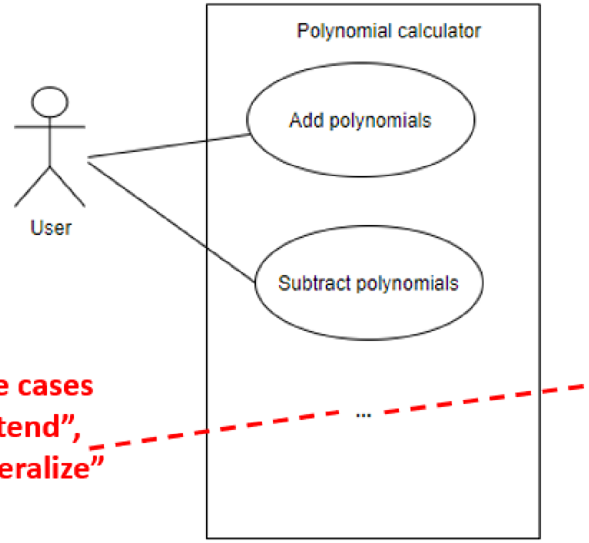


1. Clearly state the main objective and the sub-objectives required to reach it.
2. Analyze the problem and define the functional and non-functional requirements.
3. Design the solution
4. Implement the solution
5. Test the solution

Objectives

- Main objective
  - Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation to be performed and view the result.
- Sub-objectives
  - Analyze the problem and identify requirements
  - Design the polynomial calculator
  - Implement the polynomial calculator
  - Test the polynomial calculator

Analysis



Can you identify use cases connected with "extend", "include" and "generalize" relationships?

What other use cases can you identify?

Use Case: add polynomials
Primary Actor: user
Main Success Scenario:

1. The user inserts the 2 polynomials in the graphical user interface.
2. The user selects the "addition" operation
3. The user clicks on the "compute" button
4. The polynomial calculator performs the addition of the two polynomials and displays the result

Alternative Sequence: Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables)
- The scenario returns to step 1

Define requirements

Functional requirements:

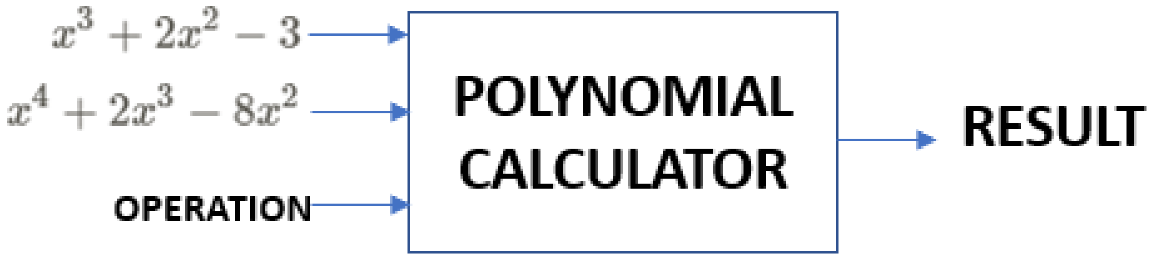
- The polynomial calculator should allow users to insert polynomials
- The polynomial calculator should allow users to select the mathematical operation
- The polynomial calculator should add two polynomials
- … what other functional requirements can you define? …

Non-Functional requirements:

- The polynomial calculator should be intuitive and easy to use by the user
- … what other non-functional requirements can you define? …

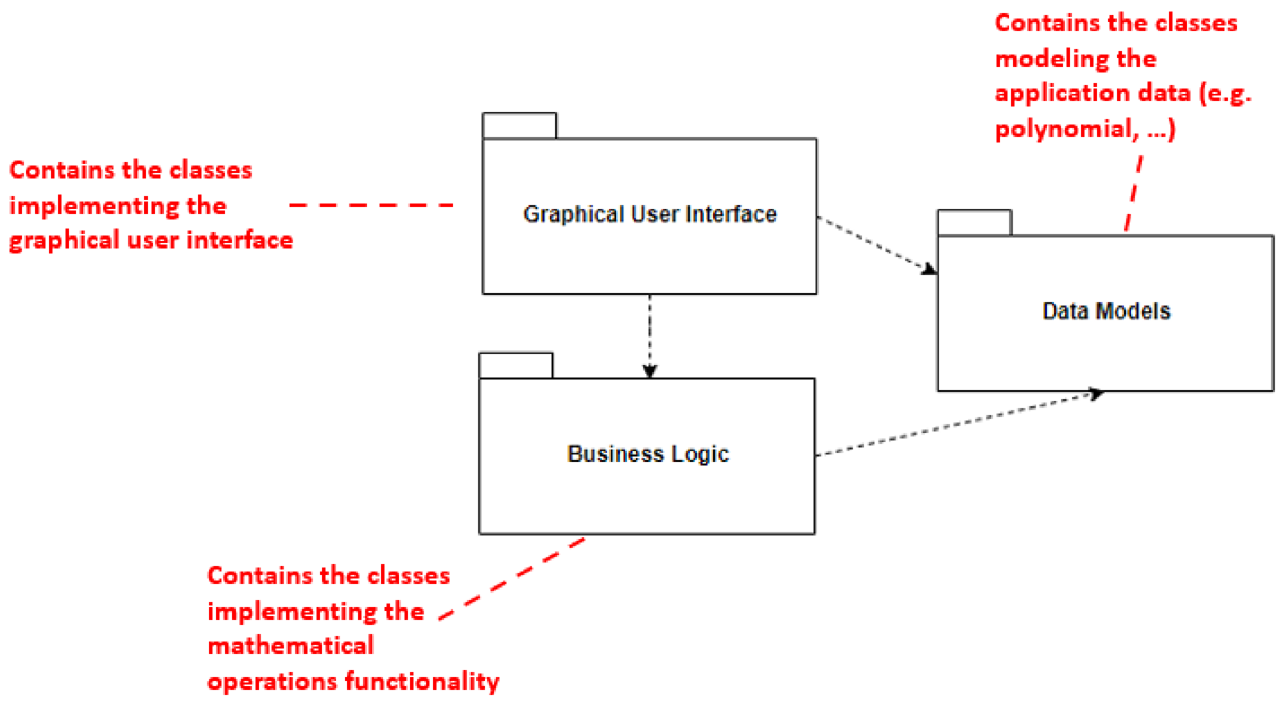
Design

Level 1: Overall system design



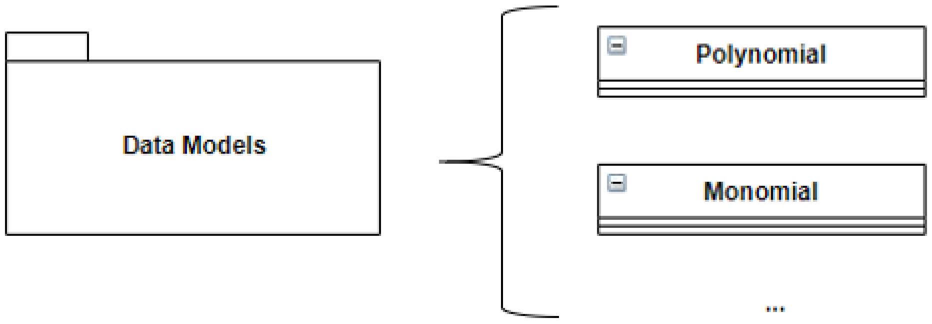
Design

Level 2: Division into sub-systems/packages



Design

Level 3: Division into classes



- What is the relation between polynomial and monomial?
- Which are the attributes of the Polynomial and Monomial classes?
- Which are the public methods of the Polynomial and Monomial classes?

... How do we devise the other packages in classes?

! When defining the classes think about **ABSTRACTION, INHERITANCE,** and **ENCAPSULATION**

Design

Level 3,4: Division into classes, Division into routines

A List-based Approach

```
Polynomial
- monomials: List<Monomial>
+ addMonomial(Monomial monomial): void
...
```

Operations Complexity

List Implementation	get	add	contains	next	remove
ArrayList	O(1)	O(1)	O(n)	O(1)	O(n)
LinkedList	O(n)	O(1)	O(n)	O(1)	O(1)

A Map-based Approach

```
Polynomial
- monomials: Map<Integer, Monomial>
+ addMonomial(Monomial monomial): void
...
```

Operations Complexity

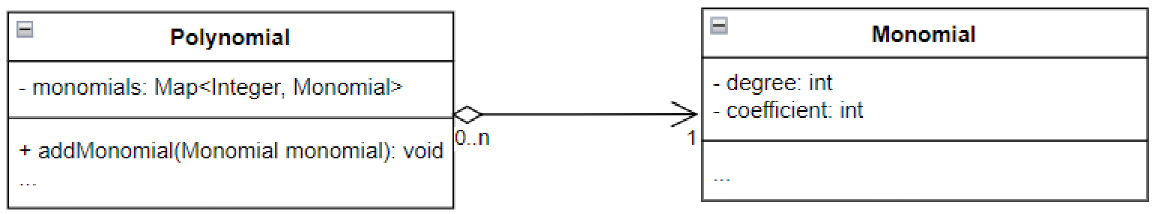
Map Implementation	get	put	containsKey	remove
HashMap	O(1)	O(1)	O(1)	O(1)
LinkedHashMap	O(1)	O(1)	O(1)	O(1)
TreeMap	O(log(n))	O(log(n))	O(log(n))	O(log(n))

More on Java Collections Complexity: [Link]

Design

Level 3,4: Division into classes, Division into routines

Try 1: A Map-based Approach

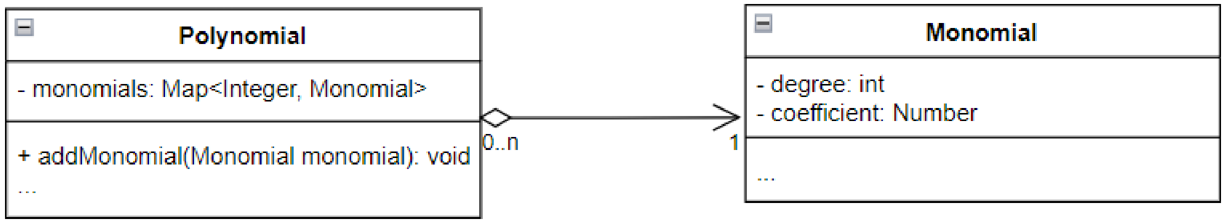


Problem with representing the double coefficients resulted from division!

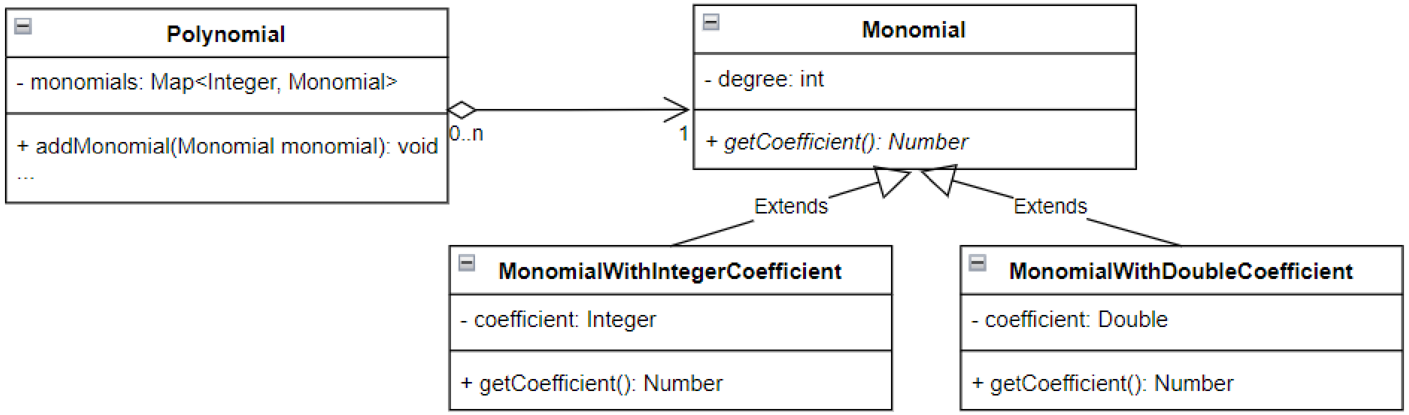
Design

Level 3,4: Division into classes, Division into routines

Try 2: A Map-based Approach



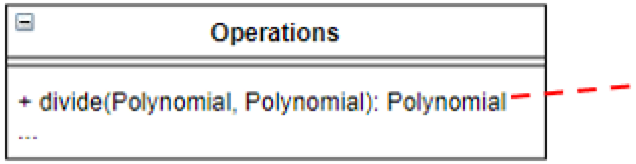
OR



! Solution that represents the double coefficients resulted from division!

Design

Level 5: Internal routine design




```
function n / d is
    require d ≠ 0
    q ← 0
    r ← n              // At each step n = d × q + r

    while r ≠ 0 and degree(r) ≥ degree(d) do
        t ← lead(r) / lead(d)       // Divide the leading terms
        q ← q + t
        r ← r − t × d

    return (q, r)
```

- **Implementation…**

Java Collections Framework

Java Collections Framework

- Unified architecture for representing and manipulating collections
  - Collection = object that contains other objects (i.e., collection elements)
    - Collection elements can be added / removed / manipulated in the collection
- Benefits
  - Reduces programming effort; increases program speed and quality; allows interoperability among unrelated APIs; fosters software reuse

Collection types

Collection type	Description	Interface
Bag	Most general form of collections; it is unordered and allows duplicate elements	Collection
Set	Does not contain duplicate elements; can be sorted	Set
List	Ordered collection of indexed elements; allows duplicate elements	List
Map	Unordered collection of associations (key, value) – the key must be unique, the value can be any entity; can be sorted	Map

Implementation Data Structure Support

Backing data structure	Targeted collection
Array	ArrayList, many Queue / Deque and Hashtables implementations
Linked List	LinkedList, LinkedBlockingQueue, ConcurrentLinkedQueue HashSet and LinkedHashSet
Hash Table	HashSet, LinkedHashSet, HashMap, LinkedHashMap, WeakHashMap, IdentityHashMap, ConcurrentHashMap
Tree	TreeSet, TreeMap, PriorityQueue, PriorityBlockingQueue

Hash table as backing data structure

- **Hash Table**
  - Backing data structure for HashSet, **LinkedHashSet**, **HashMap**, HashTable, **LinkedHashMap**, etc.
  - Used to implement an associative array (by mapping keys to values) with constant access time to its elements
    - Constant access time => no repetitive structures => direct memory access
  - The keys will be used as indexes in an array: store the pair (key, value) as

bucket[key]=value

  - The elements of the array are called **buckets**
  - **The problem with this approach is the large memory allocated and unused if the key set is sparse**
    => **Solution:** define a hash function to reduce the key set to a smaller set of size N

hash : Keys -> {1..N}

    - **The pair (key, value) will be stored as:**

bucket[hash(key)] = value

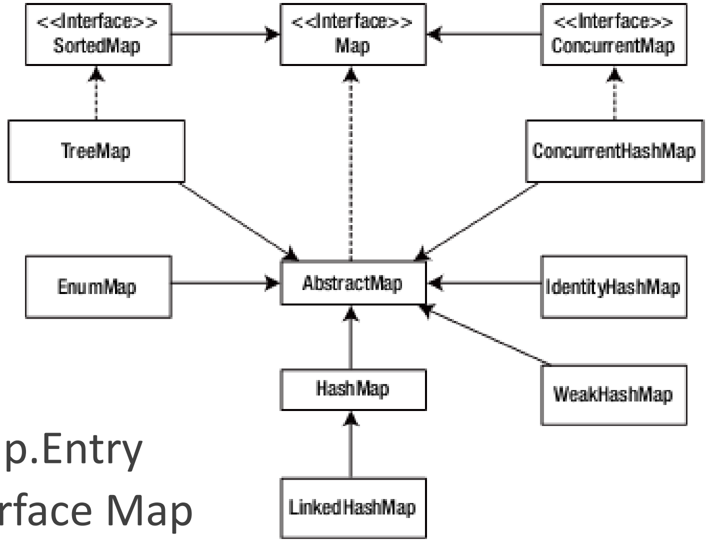
- **The hash function can lead to collisions when hash(key1) = hash(key2)**

Solved with

- **Open Addressing** : probe the next free space from the array in a given sequence
- **Chaining:** store a list in a bucket. Add all elements with the same hash value in the corresponding list

Java Map Interface

- **Java Map Interface**
  - Map
    - Object that maps keys to values
    - A (key, value) pair is an entry in the Map
    - No duplicate keys are allowed
    - One key maps to at most one value
  - Collection of Entries
    - An Entry is specified by the interface Map.Entry
      - Map.Entry - inner interface of the interface Map
  - Main Map implementations
    - Unsorted: HashMap, LinkedHashMap (inherits from HashMap)
    - Sorted: TreeMap – ordered by key
  - Iteration
    - Has no iterator method
    - **keySet()**, **entrySet()** methods return Set; **values()** method returns Collection -> Set and Collection can be iterated




```
public interface Map<K,V> {
    // Basic operations
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();
    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();
    // provides Collection Views
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();
    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

Java HashMap

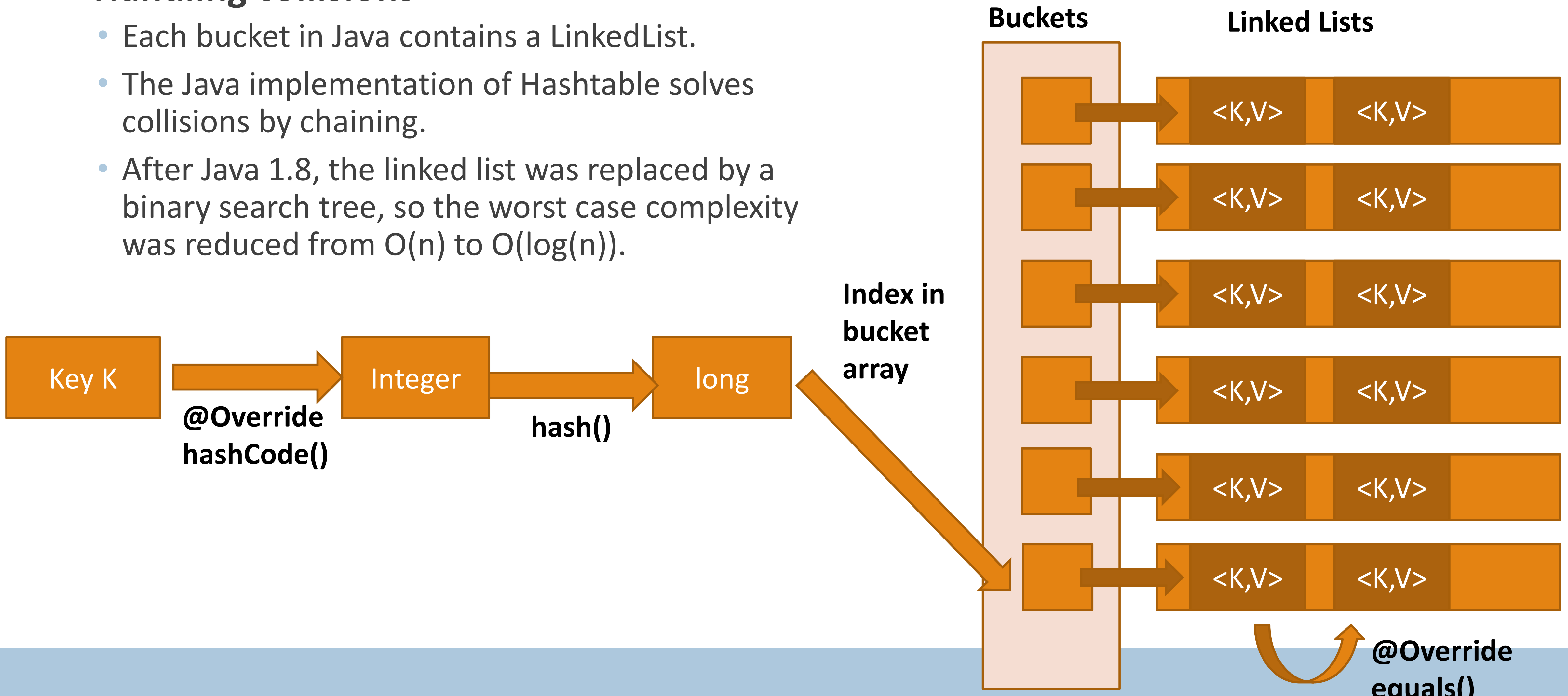
- Works on the principle of **hashing**
  - **Hashing** = assigning a unique code for any variable/object after applying any formula/algorithm on its properties
  - The **Hash function** should return the same hash code each and every time when the function is applied on same or equal objects => two equal objects must produce the same hash code
- Stores instances of the Entry class in an array:

```
transient Entry[] table;
```

```
static class Entry<K ,V> implements Map.Entry<K, V> {
  final K key;
  V value;
  Entry<K ,V> next;
  final int hash;
  ...//More code goes here
}
```

Java HashMap

- **Handling collisions**
  - Each bucket in Java contains a LinkedList.
  - The Java implementation of Hashtable solves collisions by chaining.
  - After Java 1.8, the linked list was replaced by a binary search tree, so the worst case complexity was reduced from O(n) to O(log(n)).



Java Map Interface

- **Iteration examples**

```
Map<String, String> teacherToCoursesMap = new HashMap<String, String>();
teacherToCoursesMap.put("John Doe", "Distributed Systems");
teacherToCoursesMap.put("Mary Jones", "Mathematics");
teacherToCoursesMap.put("Ann Smith", "Physics");
```

Iterate over Map.entrySet() using the for-each loop

```
for(Map.Entry<String, String> entry: teacherToCoursesMap.entrySet()){
            System.out.println("Teacher=" + entry.getKey() + "; " +
                    "Course=" + entry.getValue());
        }
```

Iterate over keys or values using the for-each loop

```
for(String teacher: teacherToCoursesMap.keySet()){
     System.out.println("Teacher=" + teacher);
}
for(String course: teacherToCoursesMap.values()){
     System.out.println("Course=" + course);
}
```

Iterate over Map.Entry<K, V> using iterators

```
Iterator<Map.Entry<String, String>> iterator = teacherToCoursesMap.entrySet().iterator();
while(iterator.hasNext()){
    Map.Entry<String, String> entry = iterator.next();
    System.out.println("Teacher=" + entry.getKey() + " , Course=" + entry.getValue());
}
```

Map Data structures comparison

Property	HashMap	HashTable	LinkedHashMap	TreeMap
Synchronization or Thread Safe	No	Yes	No	No
Null keys and null values	One null key and any number of null values	No	One null key and any number of null values	Only values
Iterating the values	Iterator	Enumerator	Iterator	Iterator
Iterator type	Fail fast iterator	Fail safe iterator	Fail fast iterator	Fail fast iterator
Interfaces	Map	Dictionary	Map	Map, NavigableMap, SortedMap
Internal implementation	Hashtable with buckets	Hashtable with buckets	Hashtable with double-linked buckets	Red-Black Tree
Get/Put average Complexity	O(1)	O(1)	O(1)	O(log(n))
Get/Put worst complexity	O(n)	O(n)	O(n)	O(log(n))
Space Complexity	O(n)	O(n)	O(n)	O(n)
Order	No guarantee that order will remain constant over time	No guarantee that order will remain constant over time	Insertion-order	Sorted according to natural ordering of the keys

Polynomial Theory

Basics of polynomial arithmetic

A *polynomial P* in an indeterminate *X* is formally defined as:

$$P(X) = a_n * X^n + a_{n-1} * X^{n-1} + \cdots + a_1 * X + \ a_0$$

where:

$c_1, c_2, \ldots, c_n$ represent the polynomial's coefficients

n represents the polynomial degree

A *monomial* is a special type of polynomial with only one term.

Consider another *polynomial Q* in the indeterminate *X* which is formally defined as:

$$Q(X) = b_n * X^n + b_{n-1} * X^{n-1} + \cdots + b_1 * X + \ b_0$$

Additional resources – polynomial arithmetic

Basics of polynomial arithmetic

Addition of two polynomials:

$$P(X) + Q(X) = (a_n + b_n) * X^n + (a_{n-1} + b_{n-1}) * X^{n-1} + \cdots + (a_1 + b_1) * X + (a_0 + b_0)$$

Example:

Consider the following two polynomials:

$$P(X) = 4 * X^5 - 3 * X^4 + X^2 - 8 * X + 1$$

$$Q(X) = 3 * X^4 - X^3 + X^2 + 2 * X - 1$$

The result of adding the two polynomials is:

$$P(X) + Q(X) = 4 * X^5 - X^3 + 2 * X^2 - 6 * X$$

Basics of polynomial arithmetic

Addition of two polynomials:

$P(X) + Q(X) = (a_n + b_n) * X^n + (a_{n-1} + b_{n-1}) * X^{n-1} + \cdots + (a_1 + b_1) * X + (a_0 + b_0)$

Example:

Consider the following two polynomials:

$P(X) = 4 * X^5 - 3 * X^4 + X^2 - 8 * X + 1$

$Q(X) = 3 * X^4 - X^3 + X^2 + 2 * X - 1$

The result of adding the two polynomials is:

$P(X) + Q(X) = 4 * X^5 - X^3 + 2 * X^2 - 6 * X$

Basics of polynomial arithmetic

Subtraction of two polynomials:

$$P(X) - Q(X) = (a_n - b_n) * X^n + (a_{n-1} - b_{n-1}) * X^{n-1} + \cdots + (a_1 - b_1) * X + (a_0 - b_0)$$

Example:

Consider the following two polynomials:

$$P(X) = 4 * X^5 - 3 * X^4 + X^2 - 8 * X + 1$$

$$Q(X) = 3 * X^4 - X^3 + X^2 + 2 * X - 1$$

The result of subtracting the polynomials is:

$$P(X) - Q(X) = 4 * X^5 - 6 * X^4 + X^3 - 10 * X + 2$$

Basics of polynomial arithmetic

Multiplication of two polynomials

To multiply two polynomials, multiply each monomial in one polynomial by each monomial in the other polynomial, add the results and simplify if necessary.

Example: Consider the following two polynomials:

$$P(X) = 3 * X^2 - X + 1$$

$$Q(X) = X - 2$$

The result of multiplying the two polynomials is:

$$P(X) * Q(X) = 3 * X^3 - X^2 + X - 6 * X^2 + 2 * X - 2 = 3 * X^3 - 7 * X^2 + 3 * X - 2$$

Basics of polynomial arithmetic

Division of two polynomials

To divide two polynomials *P* and *Q*, the following steps should be performed:

Step 1 - Order the monomials of the two polynomials *P* and *Q* in descending order according to their degree.

Step 2 - Divide the polynomial with the highest degree to the other polynomial having a lower degree (let's consider that *P* has the highest degree)

Step 3 – Divide the first monomial of *P* to the first monomial of *Q* and obtain the first term of the quotient

Step 4 – Multiply the quotient with *Q* and subtract the result of the multiplication from *P* obtaining the remainder of the division

Step 5 – Repeat the procedure from step 2 considering the remainder as the new dividend of the division, until the degree of the remainder is lower than *Q*.

Example: Consider the following two polynomials:

$P(X) = X^3 - 2 * X^2 + 6 * X - 5$

$Q(X) = X^2 - 1$

The result of dividing the two polynomials is:

$\underline{(X^3 - 2*X^2 + 6*X - 5)} : (X^2 - 1) = X - 2$

$\underline{-X^3 \quad\quad\quad + \quad X}$

$\quad - 2*X^2 + 7*X - 5$

$\quad \underline{\ \ 2*X^2 \quad\quad\quad - 2}$

$\quad\quad\quad\quad 7*X - 7$

Quotient = X – 2; Remainder = 7*X-7

Basics of polynomial arithmetic

Derivative of a polynomial

The derivative of a polynomial P is defined as follows:

$$\frac{d}{dx}(a_n * X^n + a_{n-1} * X^{n-1} + \cdots + a_1 * X + \ a_0) = n * a_n * X^{n-1} + (n-1) * \ a_{n-1} * X^{n-2} + \cdots + a_1$$

Example: Consider the following polynomial:

$$P(X) = X^3 - 2 * X^2 + 6 * X - 5$$

The derivative of polynomial P is:

$$\frac{d}{dx}(X^3 - 2 * X^2 + 6 * X - 5) = 3 * X^2 - 4 * X + 6$$

Basics of polynomial arithmetic

Integral of polynomials

The integral of a polynomial P is defined as follows:

$$\int a_n * X^n + a_{n-1} * X^{n-1} + \cdots + a_1 * X + \ a_0 = \int a_n * X^n dx \ + \int a_{n-1} * X^{n-1} dx + \cdots + \int a_1 * X dx + \int a_0 dx$$

where:

$$\int a_n * X^n dx = a * \frac{X^{n+1}}{n+1} + C$$

Example: Consider the following polynomial:

$$P(X) = X^3 + 4 * X^2 + 5$$

The integral of polynomial P is computed as:

$$\int P(X)dx = \ \int X^3 + 4 * X^2 + 5 = \int X^3 dx + \int 4 * X^2 dx + \int 5dx \quad = \frac{X^{3+1}}{3+1} + \frac{4*X^{2+1}}{2+1} + \frac{5*X^{0+1}}{0+1} + C = \frac{X^4}{4} + \frac{4*X^3}{3} + 5 * X + C$$