Object Oriented Programming and Design in Java

Session 11
Instructor: Bert Huang

Announcements

- Midterm review Monday, Mar. 8th
- Midterm exam Wednesday, Mar. 10th
- Midterm sample problems posted on courseworks

Review

- Inheritance and hierarchy
- Abstract classes
- Example hierarchies
    - Swing class hierarchy
    - awt.geom hierarchy
    - Exception hierarchy

Today’s Plan

- Java Types
  - Arrays, enums
- The Object Class
  - toString(), equals(), clone(), hashCode()
- Hash tables

Types

- Programming languages organize variables into types
- Classes are related, but don’t tell the whole story
- Types include primitives and classes
- Java is a *strongly typed* language: many compiler checks to validate type usage

Types in Java

- Types in Java are either
  - A primitive type
  - A class type
  - An interface type
  - An array type
  - The null type

Values in Java

- Values in Java are either
  - A primitive value (int, double, etc.)
  - A reference to an object of a class
  - ” ”
  - A reference to an array
  - null

Inheritance and Types

- The ideas of inheritance and hierarchy we've discussed recently apply to types
- Types can be *subtypes* of *supertypes*
- Variables of subtypes can be substituted for when a supertype variable is expected
- Liskov's substitution principle is about **types**

Rules for Java Subtypes

- S is a subtype of T if
  - S and T are the same type
  - S and T are both class types and S is a sub**class** of T
  - S is a class type, T is an interface type, and S or one of its superclasses implements interface T or one of its interfaces
  - S and T are both array types and the component type of S is a subtype of the component type of T

- S and T are the same type
- S and T are both class types and S is a sub**class** of T
- S is a class type, T is an interface type, and S or one of its superclasses implements interface T or one of its interfaces

- S and T are both array types and the component type of S is a subtype of the component type of T
- S is not a primitive and T is the type Object
- S is an array type and T is the type Cloneable or Serializable
- S is the null type and T is not a primitive Type

Primitive Types

- int, long, byte, char, float, double, boolean
- Values are stored directly in memory
- No real hierarchy; byte is **not** a subtype of int

The null Type

- Subtype of all non-primitive types
- Usually used as a placeholder before initialization
- We can check if object’s value == null

Objects

- Values are *references*: memory locations
- == will compare references, not values
- Data is stored as primitives or in the structure of references
- Objects’ types are defined by classes

Arrays

- Arrays in Java are types (String [] args)
- “S and T are both array types and the component type of S is a subtype of the component type of T”
- Is int a subtype of int []? No
- Is MouseAdapter [] a subtype of MouseListener []? Yes

Multidimensional Arrays

- Since arrays are variables of the array type, we can have arrays of arrays
    - Integer [][] grid;
- This is a subtype of Number [][], but not hierarchically connected to Integer []

enum

- Java provides a way to create special class types called *enumerated types*
- These are types that have a few possible values, but there is no order or numerical meaning to the values
  - e.g., BorderLayout.NORTH, SOUTH, EAST, WEST
- Instead of constants that a client can then read as meaningless int values, use enum type

enum Usage

- public enum Location { NORTH, SOUTH, EAST, WEST };
- Clients can instantiate Location objects, or use constants Location.NORTH, etc.
- The special syntax is sugar for "extends Enum"

java.lang.Object

- All class variables extend the base Java class, java.lang.Object
- Object contains a few implemented methods:
  - String toString()
  - boolean equals(Object other)
  - Object clone()
  - int hashCode()

toString()

- Returns String representation of the Object
- mportant in Java because it is used automatically with the + operator on Strings
- The default returns the name of the class and the *hash code* in hexadecimal
- Usually, you should override with something more useful

equals()

- Returns whether parameter is “equal” to this
- Should override with useful definition of equality. Must be
  - Reflexive (x.equals(x) always true)
  - Symmetric (x.equals(y) == y.equals(x))
  - Transitive (x.equals(y) & y.equals(z) means x.equals(z))
- Default is the actual == operation

clone()

- Clone is meant to be used when you want an actual copy of an Object instead of another reference
- (x.clone() != x) && (x.clone().equals(x))
- Default clone() copies all fields
- clone() is a protected method by default and can only be used if your subclass implements the Cloneable interface

The Cloneable Interface

- Tagging interface; contains no methods
- But Object uses it to check that calls to clone() are only on Cloneable objects
  - otherwise throws `CloneNotSupportedException`
- Must be careful; copying fields may still share common aggregated objects

hashCode()

- Returns a int representing the Object
- Must be consistent with equals()
  - if x.equals(y),
    then x.hashcode() == y.hashcode()
  - but hashcodes can be equal for different objects (this is unavoidable)
- Must be overridden to be useful

Hash Tables

- A hash table fixes a major complaint about arrays and lists:
  - Why do I have to look up elements by integer indices?
- e.g., “index” values by String, A[“John”]
- Refer to the “index” as the *key*

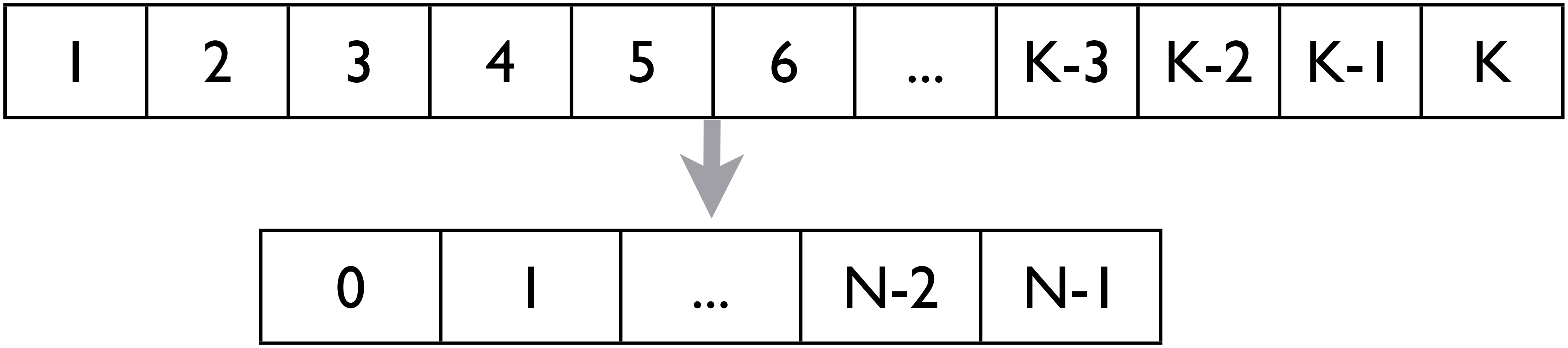
Initial Intuition

- If we have infinite memory, we can enumerate all possible keys 1 through K
- Create an array with K entries
- Insert, delete, search are just array operations

1	2	3	4	5	6	...	K-3	K-2	K-1	K

Hash Functions

- A **hash function** maps any key to a valid array position
  - Array positions range from 0 to N-1
  - Key range possibly unlimited



1	2	3	4	5	6	...	K-3	K-2	K-1	K

0	1	...	N-2	N-1

HashTable

- HashTable<Key, Value>()
- Stores values according to the key's hashcode()
  - Value get(Key k)
  - Value put(Key k, Value v)
  - boolean contains(Value v)
  - boolean containsKey(Key k)

Bonus: More Hashing Details

- For integer keys, (key mod N) is the simplest hash function
- In general, **any** function that maps from the space of keys to the space of array indices is valid
- but a good hash function spreads the data out evenly in the array
- Collisions will happen, but hopefully rarely.
    - Handle by storing in a list or in a systematic way in other array locations

Reading

- Horstmann Ch. 7.1-7.4