B07 Oct 4 Lec 1 Notes

Procedural Paradigm

- Focuses on designing methods
- Data and operations on the data are separate.

Object-oriented paradigm

- Couples methods and data together into objects.
- Organizes programs in a way that mirrors the real world.
- A program can be viewed as a collection of cooperating objects
- Makes programs easier to develop and maintain
- Improves Software reusability.

Inheritance

- Helps avoid redundancy
- Inheritance allows developers to
  - Define a general class (or superclass) e.g. Person
  - Extend the general class to a specialized class (or subclass) e.g. employee

Casting Objects

- We can cast an instance of a sub class to a variable of a superclass (known as upcasting)
  - e.g. `Person p = new Person();`
- When casting an instance of a superclass to a variable of its subclass (downcasting), explicit casting must be used.
  - e.g. `Person p = new Employee(); Employee e = (Employee) p;`
  - It is good practice to use the `instanceof` operator to ensure that the object is an instance of another object before attempting a casting.

Overloading

- Defining methods have the same name but different signatures.

Overriding

- Defining a method in the subclass using the same signature and the same return type as in its superclass.
- The `@Override` annotation helps avoid mistakes.

Super

- Can be used to invoke a superclass constructor
  - Must be the first statement of the subclass constructor.
  - A constructor may invoke an overloaded constructor or its superclass constructor. If neither is invoked explicitly, the compiler automatically puts `super()` as the first statement in the constructor.
- Can be used to invoke a superclass method.
  - Syntax: `super.methodName(parameters)`

Object Class

- Every Java class has Object as superclass

Polymorphism

- Every instance of a subclass is also an instance of its superclass, but not vice versa.
- An object of a subclass can be used whenever its superclass object is used.

Dynamic Binding

- A method can be implemented in several classes along the inheritance chain.
- The JVM dynamically binds the implementation of the method at runtime, decided by the actual type of the variable.
  - e.g. `Object x = new Point(1,2);`
    `System.out.println(x);`
- Dynamic binding works as follows:
  - Suppose an object x is an instance of classes $C_1, C_2, \cdots, C_{n-1}$, and C_n, where C_1 is a subclass of C_2, C_2 is a subclass of $C_3, \cdots$.
  - If x invokes a method p, the JVM searches for the implementation of the method p in $C_1, C_2, \cdots, C_{n-1}$, and C_n in this order until its found. Once an implementation is found, the search stops and the first-found implementation is invoked.

Encapsulation

- The details of implementation are encapsulated and hidden from the user.
- Modules communicate only through their APIs and are oblivious to each others' inner workings.

Abstract Classes

- Cannot be instantiated using the new operator.
- Usually contain abstract methods that are implemented in concrete subclasses.
- A class that contains abstract methods must be defined as abstract.
- If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be defined as abstract.

Interfaces

- An interface can be used to define common behaviour for classes (including unrelated classes)
- Contains only constants and abstract methods.

Generics

- Enable type parametrization
  - Generic Interfaces
  - Generic Classes
  - Generic Methods
- Example: ArrayList Class
  - Generic types must be reference types.
  - Generic types could be bounded using the extends keyword.

Generics (The Comparable Interface)

- Comparable is a generic interface
  - Defines the compareTo method for comparing objects.
- The compareTo method determines the order of the calling object with t and returns a negative integer, zero, or a positive integer if the calling object is less than, equal to, or greater than t.
- Many classes implement Comparable (e.g. String, Integer).

Generics (The HashSet Class)

- Generic class that can be used to store elements without duplicates.
  - No two elements e_1 and e_2 can be in the set s.t. e_1.equals(e_2) is true.

Generics (The LinkedHashSet Class)

- Elements of a HashSet are not necessarily stored in the same order they weve added.
- LinkedHashSet is a subclass of HashSet

Exception

- An exception is an object that represents an error or a condition that prevents execution from proceeding normally.
- Exceptions are represented in the Exception Class, which describes errors caused by the program and by external circumstances.
- Developers can create their own exception classes by extending Exception.
- In Java, runtime exceptions are represented in the RuntimeException class. Subclasses include:
  - ArrayIndexOutOfBoundsException
  - NullPointerException
- RuntimeException and its subclasses are known as unchecked exceptions.
- All other exceptions are know as checked exceptions.
  - The compiler forces the programmer to check and deal with them in a try-catch block or declare them in the method header.
- Declaring exceptions
  - Every method must state the types of checked exceptions it might throw using the throws keyword in the header.
- Throwing exceptions
  - A program that detects an error can create an instance of an appropriate exception type and throw it using the throw keyword.

Equals method

- When you override the equals method, you must adhere to its general contract.
  - Reflexive: For any non-null reference value x, x.equals(x) must return true.
  - Symmetric: For any non-null reference values x and y, x.equals(y) must return true iff y.equals(x) returns true.
  - Transitive: For any non-null reference values x,y,z. if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) must return true.
  - Consistent: For any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false.
  - For any non-null reference value x, x.equals(null) must return false.