Main

Data structures / types

primitive: byte/short/int/long boolean/char float/double

not primitive: Object String List System ArrayList

class - a template / language construct that encapsulates data/behaviour object - instance of a class method - action/behaviour that encapsulates code

a. array - sequence of items, fixed size, cont piece of memory b. list (linked) - sequence of items, dynamic size, 1 -> 2 -> 3.. no cont piece of memory

c. dictionary/map -

`[key3][val3] [key1][val1] [key2][val2]`

d. queue - FIFO / stack - LIFO e. graph - a collection of nodes with a collection of edges

---

Abstractions / Implementation

abstract class - can "extend" only one interface - can "implement" multiple

---

UI - JavaFX / Swing (AWT)

Model-View-Controller (MVC) Model - (business) logic View - visual design Controller - link / glue between Model and View

MVP, MVVM, etc.

Test categories: Unit, Integration, System

unit: single functionality integration: multiple functionalities system: the whole software program using user scenarios

Test approaches: black-box, white-box

black-box: don't know how code works, know what it should do white-box: know how code works + know what it should do

---

Test-driven Development method: red-green-refactor

context: simple calculator i. red - write test first,

add(1, 2) = 3, sub(1, 2) = -1, mul(1, 2) = 2, div(1, 2) = 0.5

```
@Test
public void testAdd() {
    var calc = new Calculator();
    var result = calc.add(1, 2);
    assertThat(result, is(3));
}
```

ii. green - write the simplest main code that passes the test from i. iii. refactor - clean up main code

---

Design patterns:

Design pattern - a way of structuring code for readability purposes

- a common set of solutions to a common set of problems

Builder - for constructing types with a large number of parameters Factory - for creating objects without exposing concrete types Observer - for observing changes in values / objects Singleton - for cases where only a single instance of a type is needed Dependency Injection - to make code future-proof; allows providing different dependencies (i.e. they are not hardcoded)

```
public final class SingletonClass {
    private static final SingletonClass singleton = new SingletonClass();

    private SingletonClass() {}

    public static SingletonClass getSingleton() {
        return singleton;
    }
}
```

OR

```
public enum SingletonClass {
    INSTANCE;
}
```

Multi-threading

Thread - a worker that can run a task, multiple threads allow execution of multiple tasks

Use cases:

- UI updates
- Playing music in the background in a shopping app
- Connecting to a database while animating the UI

```
Thread t1 = new Thread(() -> {
    // code to run in background
});
t1.start();
```

Class diagrams

- define classes / units in terms of public (+) / private (-) fields and methods

Interaction diagrams

- how the process (methods/data) flows from one class / unit to another