OBJECT INTERACTION

CITS1001 week 3
Fundamental concepts

- Coupling and Cohesion
- Internal/external method calls
- null objects
- Chaining method calls
- Class constants
- Class variables

Reading: **Chapter 3** of *Objects First with Java - A Practical Introduction using BlueJ*, © David J. Barnes, Michael Kölling
Concepts (1)

- abstraction
- modularization
- classes define types
- class diagram

- object diagram
- object references
- object types
- primitive types
A digital clock

11:03
Abstraction and modularization

- **Abstraction** is the ability to ignore details of the parts of a problem, to focus attention on its higher levels.

- **Modularization** is the process of dividing a whole into well-defined parts, which can be built and examined separately, and which interact in well-defined ways.
Modularizing the clock display

11:03

One 4-digit display?

Or two 2-digit displays?

11 03
Implementation - NumberDisplay

```java
public class NumberDisplay {
    private int limit;
    private int value;

    Constructor and methods omitted.
}
```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;

    Constructor and methods omitted.
}
Object diagram

Dynamic view at *runtime* (when the system is running)
Object diagram

- Objects exist at run-time
- The **object diagram** shows the objects and their relationships at one moment in time during the execution of an application
- It gives information about objects at runtime and presents the **dynamic** view of a program
Class diagram

ClockDisplay \textit{depends on} NumberDisplay

ClockDisplay \textit{makes use of} NumberDisplay
Classes define types

```java
private NumberDisplay hours;
```

- A class name can be used as the type for a variable
- Variables that have a class as their type can store objects belonging to that class
Class diagram

• Classes exist at compile time
• The **class diagram** shows the classes of an application and the relationships between them
• It gives information about the source code and presents the **static** view of a program
Primitive types vs. object types

```java
SomeObject obj;

int i;
```

object type

primitive type

32
Quiz: What is the output?

•  int a;
   int b;
   a = 32;
   b = a;
   a = a + 1;
   System.out.println(b);

•  Person a;
   Person b;
   a = new Person("Everett");
   b = a;
   a.changeName("Delmar");
   System.out.println(b.getName());
Primitive types vs. object types

```
ObjectType a;

 ObjectType b;

int a;

 int b;
```

```
b = a;
```

```
32
```

```
32
```
The modulo operator

- The 'division' operator (/), when applied to int operands, returns the result of an integer division.
- The 'modulo' operator (%) returns the remainder of an integer division.
- E.g., generally:
  \[ 17 \div 5 \text{ gives result } 3, \text{ remainder } 2 \]
- In Java:
  \[
  17 \div 5 == 3 \\
  17 \mod 5 == 2
  \]
Quiz

• What is the result of the expression
  \[ 8 \mod 3 \]

• For integer \( n \geq 0 \), what are all possible results of:
  \[ n \mod 5 \]

• Can \( n \) be negative?
public NumberDisplay(int rollOverLimit) {
    limit = rollOverLimit;
    value = 0;
}

d public void increment() {
    value = (value + 1) % limit;
}
Concepts (2)

• abstraction
• modularization
• classes define types
• class diagram

• object diagram
• object references
• object types
• primitive types
Objects creating objects

```java
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        ...
    }
}
```
Objects creating objects

in class ClockDisplay:

hours = new NumberDisplay(24);

actual parameter

in class NumberDisplay:

public NumberDisplay(int rollOverLimit);

formal parameter
ClockDisplay object diagram
Method calling

```java
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {
        // it just rolled over!
        hours.increment();
    }
    updateDisplay();
}
```
External method call

• For calling a method on another object

• external method calls

  `minutes.increment();`

  `object . methodName ( parameter-list )`

where public void increment() { ... }

If increment() had been a private method we would not have been able to invoke it.
Internal method call

• For calling a method on our own object

• internal method calls

    `updateDisplay();`

• No variable name is required.

• `this`
  • could be used as a reference to the invoking object, but not used for method calls.
Internal method (helpers)

/**
 * Update the internal string that represents the display.
 */
private void updateDisplay()
{
    displayString =
        hours.getDisplayValue() + "":" +
        minutes.getDisplayValue();
}
Method calls

• NB: A method call on another object of the same type would be an external call.
• ‘Internal’ means ‘this object’.
• ‘External’ means ‘any other object’, regardless of its type.
null

- null is a special Object in Java
- All Object variables (of any class) are initially null
- Variables can be tested for null

private NumberDisplay hours;

if(hours != null) {
    //... nothing to show
} else {
    // ... display the hours
}

- Variables can be assigned to null - losing the reference to anything they were previously holding.

public void forgetHours() {
    hours = null;
}
Anonymous objects

- Objects are often created and handed on elsewhere immediately:

  ```java
  Lot furtherLot = new Lot(...);
  lots.add(furtherLot);
  ```

- We don’t really need `furtherLot`:

  ```java
  lots.add(new Lot(...));
  ```
Chaining method calls

- Methods often return objects.
- We often immediately call a method on the returned object.

```java
Bid bid = lot.getHighestBid();
Person bidder = bid.getBidder();
```

- We can use the anonymous object concept and chain method calls:

```java
lot.getHighestBid().getBidder()
```
Chaining method calls

• Each method in the chain is called on the object returned from the previous method call in the chain.

```java
String name =
    lot.getHighestBid().getBidder().getName();
```

- Returns a **Bid** object from the **Lot**
- Returns a **Person** object from the **Bid**
- Returns a **String** object from the **Person**
Concept summary

- object creation
- overloading
- internal/external method calls
- debugger
Review (1)

- Abstraction
  - ignore some details to focus attention on a higher level of a problem

- Modularisation
  - Divide a whole into well defined parts that can be built separately and that interact in well-defined ways

- Classes define types
  - A class name can be used as the type for a variable. Variables that have a class as their type can store objects of that class.
Review (2)

- **Object diagram**
  - Shows the objects and their relationships at one moment during the execution of an application

- **Object references**
  - Variables of object types store references to objects

- **Primitive type**
  - The primitive types of Java are non-object types. The most common primitive types are `int`, `boolean`, `char`, `double` and `long`.

- **Object creation**
  - Objects can create other objects using the `new` operator
Review (3)

• Overloading
  • A class can contain more than one constructor or more than one method with the same name. In this case, each must have a distinctive set of parameter types.

• Internal method call
  • Methods can call other methods of the same class.

• External method call
  • Methods can call methods of other objects using dot notation