Working with objects

Object basics

- Objects (instances) are self-contained elements of a program.
- Objects contain:
  - Attributes - In Java these are variables
  - Behavior - In Java these are methods

Using the “new” operator

- Objects are created from classes which serve as their templates.

  Examples:

  - `String name = new String();`
  - `URL address = new URL(“http://www.java21days.com”);`
  - `VolcanoRobot robbie = new VolcanoRobot();`
- **Constructor** - special method used to create and initialize object instances.
- One class can have one or more constructors with different arguments.

Example: TokenTester

Working with instance variables

- Reading instance variables:
  - **Dot notation** is a way of referring to an object's instance variables using the dot "." operator.

    Examples:
    - `float total = myCustomer.orderTotal;`
    - `boolean onLayaway = myCustomer.orderTotal.layaway;`
- Changing instance variables:

  Example:

  ```
  myCustomer.orderTotal.layaway = true;
  ```

Example: PointSetter

Working with class variables

Defining class variables:

- Use the **static** keyword before the variable to make it a class variable.

  Example:

```
class FamilyMember {
    static String surname = “Mendoza”;
    String name;
    int age;
}
```

Working with class variables

Reading class variables:

Example:

```
FamilyMember dad = new FamilyMember();
System.out.println(“Family’s surname is: “ + FamilyMember.surname);
```

This works too but should be avoided:

```
System.out.println(“Family’s surname is: “ + dad.surname);
```

Working with methods

Methods are invoked using the dot notation

Example: StringChecker

Working with class methods

- Used for general utility methods.
- They operate on the objects provided as arguments, not on the instance of the class where they're defined.
- Examples:
  - `String s = String.valueOf(550);`
  - `int higherPrice = Math.max(firstPrice, secondPrice);`
- This works too but should be avoided:
  - `String s = “item”;`
  - `String s2 = s.valueOf(550);`

Object references

- A **reference** is an address that indicates where an object's variables and methods are stored.
- Assigning an object to a variable or passing an object as an argument does not create a copy of the object. Instead, a reference is used.

Example: ReferenceTester

Casting

- Java is strongly typed languages; converting primitives or objects from one type to another requires **casting**.
- Casting is the process of producing a new value that has a different type than its source.
- There are three situations that can occur:
  - Casting between primitive types, such as int to float or float to double
  - Casting from an instance of a class to an instance of another class, such as Object to String
  - Casting primitive types to objects and then extracting primitive values from those objects

Casting primitive types

- Mostly used with numeric types.
- Conversion from a "smaller" to a "larger" type is easy and requires no explicit casting.
- Conversion from a " larger" to a "smaller" type requires an explicit cast due to possible lost in precision.
- Example:

```
long l = 100;
int i = (int) l;
l = i;
```

- Example:

```
int result = (int) (x / y);
```

Casting objects

Restriction: the source and destination classes must be related by inheritance i.e. one class must be a subclass of the other.

Example:

```
Employee emp = new Employee();
VicePresident veep = new VicePresident();
emp = veep; // no cast needed for upward use
veep = (VicePresident) emp; // must cast explicitly
```

Converting primitive types to objects and vice versa

- Classes alternative to primitive types: Float, Boolean, Byte, Integer, Character…
- Casting is not supported; instead constructors or class methods can be used to convert them.
- Examples:
  - `Integer dataCount = new Integer(7801);`
  - `int newCount = dataCount.intValue();`

Converting primitive types to objects and vice versa

This conversion is made easier by **autoboxing** and **unboxing** where Java does the conversion.

Example:

```
Float f1 = new Float(12.5F);
Float f2 = new Float(27.2F);
System.out.println("Lower number: " + Math.min(f1, f2)); //
Math.min(float, float)
```

Comparing object types

- Most operators work on primitive types ("==", "!=", "<", …)
- Actually, "==" and "!=" work but not as you would expect; the check if left and the right operator refer to the same object, not if they have the same value!
- In Java, two objects are compared if they're equal using the equals() method.

Example: EqualsTester

Determining the class of an object

The **getClass()** method can be used to determine the class which was used to create an object.

Example:

```
String name = key.getClass().getName();
```

Determining the class of an object

The **instanceof** operator can be used to determine if an object instance is of a particular class.

Example:

```
boolean check1 = “Texas” instanceof String; // true
```

Example:

```
Point pt = new Point(10, 10);
boolean check2 = pt instanceof String; // false
```

Exercises

Exercise: DateParser

Create a program that turns a birthday in MM/DD/YYYY format (such as 12/04/2007) into three individual strings which it then displays.

Exercise: BoxVolume

Create a class with instance variables for height, weight, and depth, making each an integer. Create a Java application that uses your new class to create two instances to which it then sets values for the attributes, and finally displays the values of the object that has the greater volume.