COMP 251

Java review
Types

Primitive types: byte, short, int, long, float, double, boolean, char

Object types: String, Cat, Dog, int [], String [], etc.
Types: primitive types store values

```java
int[] a = 5;
int b = a;
b++;
// value of a?
```
Types: primitive types store values

```java
int[] a = 5;
int b = a;
b++; // value of a?
```

Answer: 5
Types: object types store references

```java
int[] c = {5, 5};
int[] d = c;
d[0]++;
// value of c[0]?
```
Types: object types store references

```java
int[] c = {5, 5};
int[] d = c;
d[0]++;
// value of c[0]?
```

Answer: 6
Methods

- Piece of code that does some computations
- Takes some parameters in (or none)
- Outputs something out (or not)
- Can be called from anywhere (if it is public)
Parameter passing: primitive types

```java
public static void main (String[] args) {
    int a = 5;
    doSomething(a);
    // value of a?
}

static void doSomething (int a) {
    a++;}
```
Parameter passing: primitive types

```java
public static void main (String[] args) {
    int a = 5;
    doSomething(a);
    // value of a?
}

static void doSomething (int a) {
    a++;}
```

Answer: 5
Parameter passing: object types

```java
public static void main (String[] args) {
    int[] c = {5, 5};
    doSomethingElse(c);
    // value of c[0]?
}

static void doSomethingElse (int[] c) {
    c[0]++;
}
```
Parameter passing: object types

```java
public static void main (String[] args) {
    int[] c = {5, 5};
    doSomethingElse(c);
    // value of c[0]?  
}

static void doSomethingElse (int[] c) {
    c[0]++;
}
```

Answer: 6
Parameter passing: object types

```java
public static void main (String[] args) {
    int[] c = {5, 5};
    doSomethingElse(c);
    // value of c[0]?
}

static void doSomethingElse (int[] c) {
    c = new int[3];
    c[0]++;
}
```
Parameter passing: object types

```java
public static void main (String[] args) {
    int[] c = {5, 5};
    doSomethingElse(c);
    // value of c[0]?
}

static void doSomethingElse (int[] c) {
    c = new int[3];
    c[0]++;
}
```
Answer: 5
Parameter passing: summary

Parameters are passed by value

- The method gets a copy of the parameters passed
- If it is a primitive type, the method gets a copy of the value, so the it can't change the original
- If it is a non-primitive type, the method gets a reference to the original! So it can modify the original
Classes, objects

Class

- A class is a prototype from which objects are created.
- It describes the state (variables) and behaviors (methods) that are common to all objects of this type

Object

- A specific variable of that type (instance of the class)
Classes, objects: instantiating and using objects

- Each class is stored in its own file (in general)

- The Driver / Main class contains the main() method, which is the entry point of the program

```java
3 class Book {
4     private String title;
5     private double price;
6 }

8 public class Driver {
9     public static void main (String[] args) {
10        Book myBook;
11        myBook = new Book();
12     }
13 }
```

This calls a method called the constructor… more on this soon
Classes, objects: instance variables

```java
class Book {
    private String title;
    private double price;

    public void setTitleToDefault() {
        title = "abc";
    }
}
```

What happens if we change the setTitleToDefault method to this?

```java
public void setTitleToDefault() {
    String title;
    title = "abc";
}
```
Classes, objects: instance variables

This won’t work:

```java
class Book {
    private String title;
    private double price;

    public void setTitle(String title) {
        title = title;
    }
}
```

Use `this` keyword

```java
public void setTitle(String title) {
    this.title = title;
}
```
Classes, objects: instance variables summary

- To access an instance variable within the class, we do:
  - `nameOfVariable`

- What if a local variable or a parameter have the same name?
  - The local variable or parameter will be used
  - Solution: use `this` keyword
  - `this` refers to the current object
Classes, objects: constructor

- A method that is called automatically when an instance of a class is created
- Usually used to initialize the instance variables
- Has the same name as the class
- If you don’t make one, Java provides a default no-argument constructor

```java
public class Book {
    private String title;
    private double price;

    public Book (String title, double price) {
        this.title = title;
        this.price = price;
    }
}
```
Classes, objects: static members

- We declare members to be static if we want them to be shared by all objects of a class
- Static members belong to the class, not to the specific instances of it
Classes, objects: static variables

With non-static variables:

- each object has its own copy of the variable

With static variables:

- Only one copy of the variable exists to be shared by all objects of the class
- All objects of the class can read and change a static variable
- Useful to share info among objects of the same class
  - ex: constants, nb of objects created, ...
Classes, objects: static methods

Invoked through the name of the class

```
int totalNumOfBooks = Book.getTotalNumberOfBooks();
```

A static method can be invoked even if no object of that class exists!

- E.g. the main() method
- Note that instance (non-static) variables do not exist until an object exists

Can static methods access...

- non-static variables and methods? No
  - example: main can only call static methods
- static variables and methods? Yes

Can static methods use this reference?
Wrapper classes

Wrap a primitive value into a class object

Why?

- if we want to modify the arguments passed into a method (because primitive types are passed by value)
- Data structures in the Collection framework, e.g. LinkedList only work with objects

Java automatically converts primitive values into the corresponding wrapper class (boxing) and vice versa (unboxing)

```
int x = 20;  // primitive type
Integer y = new Integer(20);  // reference type
```
Generics: creating objects of a generic class

Generic classes: allow to have a single class that works with different types.

Some of the data structures provided by the Java utility package (Collections) use generics.

For example in `LinkedList<T>`
- `<T>` is a generic
- can be read as "of type T"

```java
LinkedList<Integer> myList = new LinkedList<Integer>();
myList.add(3);
```
Stuck?

1. Check the Java documentation: [https://docs.oracle.com/javase/7/docs/api/](https://docs.oracle.com/javase/7/docs/api/)
2. Google your problem, someone already had it (99% of the times :)
3. CSUS helpdesk: [https://mcgill-csus.ca/](https://mcgill-csus.ca/)
4. Office hours