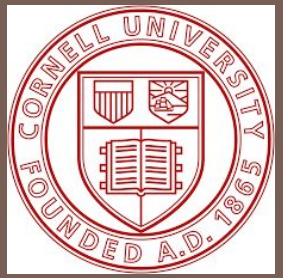
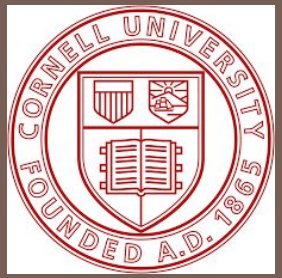
Object-oriented programming and data-structures



CS/ENGRD 2110 SUMMER 2018



Lecture 2: Objects

http://courses.cs.cornell.edu/cs2110/2018su

Lecture 1 Recap

- Primitive Types in Java
- Functions/Procedures
- Basic Control Flow Structures
- Local Variables

Lecture 2 Objects (finally …)

- Objects
- How to define an object.
- How to use an object.
- Constructors
- Pass-by-value, pass-by reference

Why object-oriented?

- Primitive types become restrictive
- May want to group related information together
  - Ex: a Date consists of
    - A day (int), a month (String), a year (int)
- Might not want to just store data, but also associated functions
  - Ex: A function that prints the date in British or American style.
- Use these complex types as building blocks for your program

Classes

- **Definition** Classes describe the blueprint/template of different concepts (Date, Person, Animal, etc.)

- Classes group conceptually related **state** (as **fields**) and **behaviour** (**methods**)
  - **Fields** Variables that belong to a class
  - **Methods** Functions or procedures that belong to a class

- **Objects** represent distinct **instances** of a class
  - Person natacha;
  - Person chris;
    - **Object** natacha is an **instance** of class **Person**

Defining a class

- Declare **state** and define **methods** it contains

Defining a class

- Declare **state** and define **methods** it contains

```
/** Definition of what class is for **/

class Date {

        String month;
        int day;
        int year;

        void printDateUK() {...}
        void printDateUS {...}

}
```

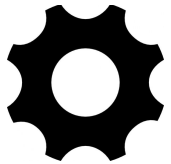
State

Behaviour

Class definition **Date** goes in its own file named **Date.java**

On your hard drive, have separate directory for each Java project you write; put all class definitions for program in that directory. You'll see this when we demo.

Commenting



- Every field should have a comment describing what it represents and what valid inputs are
- A class should have comments describing its purpose
- Method functionality should also be described
  - Methods have a precondition, and a postcondition
    - Ex: in a setDay(int day) method, precondition is that day to be below 31.
- JavaDS describes what we expect (you'll see in homework)

Creating instances/objects

- Objects are created in three steps
  - They are **declared**: give a variable name and an object type
    - Date dateOfBirth;

Creating instances/objects

- Objects are created in three steps
  - They are **declared**: give a variable name and an object type
    - Date dateOfBirth;
  - They are **instantiated:** a memory location is created for that object and fields are assigned default values
    - Date dateOfBirth = **new**

Creating instances/objects

- Objects are created in three steps
  - They are **declared**: give a variable name and an object type
    - Date dateOfBirth;
  - They are **instantiated:** a memory location is created for that object and fields are assigned default values
    - Date dateOfBirth = **new**
  - They are **initialised:**
    - Date dateOfBirth = new Date();
    - Call to **constructor** initialises the object;

Constructors

- Constructor
  - Method called when object is constructed
  - Initialize fields of a new object so that its class invariant is true
- Constructor has the **same name** as the class, and **no return type**
- Every class has an (implicit?) **default** constructor
- Classes may have multiple constructors**.**

Constructors

```
/** Definition of what class is for **/
class Date {
        String month;   int day;    int year;
         /** Constructor: instance with pMonth, pDay, pYear. Precondition, pMonth in Jan-Deb, pDay in 0/31, pDay in .
*/
        Date(String pMonth, int pDay, int pYear) {
                month = pMonth;
                day = pDay;
                year = pYear;
        }
        Date(int pMonth, int pDay, int pYear) {
                month = convertToString(pMonth);
                day = pDay;
                year = pYear;
        }
        void printDateUK() {...}
        void printDateUS {...}
}
```

Constructors also allow to check whether input to fields is consistent with precondition without repetition. Ex: replacing day with day = checkInRange(pDay)

Using Objects And Classes

- Classes can be used as part of building blocks for more complex types
  - Compose classes easily
  - class Person { String name ; Date dob … }

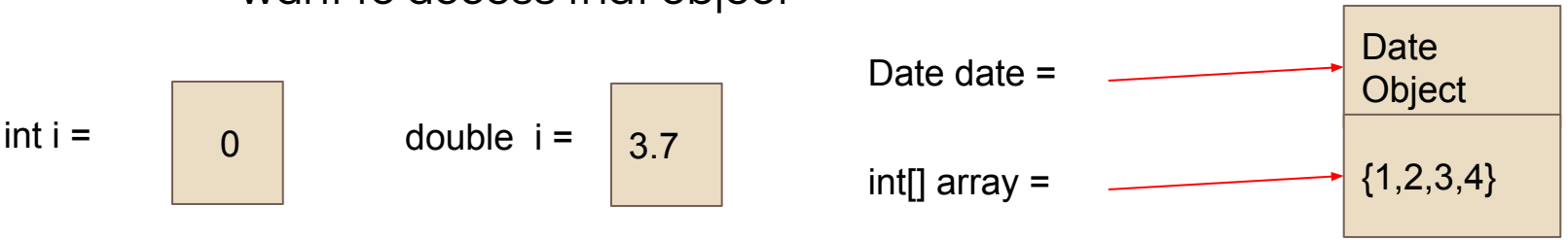
Using Objects And Classes

- Classes can be used as part of building blocks for more complex types
  - Compose classes easily
  - class Person { String name ; Date dob … }

- Fields in a class are accessed through an **instance** of that class
  - Given an instance **Date date**, access field **month** by date.month
- Methods in a class are accessed through an **instance** of that class
  - Given an instance **Date date**, call method **printDateUK**
    date.printDateUK()

References

- Recall: Java has **primitive types** and **class types**
- When declare a primitive type, return that type directly
- When declare and create an object via the **new** keyword, return a reference to that object
  - Can view it as a **name** for the object that we can look up whenever want to access that object



References - Consequences

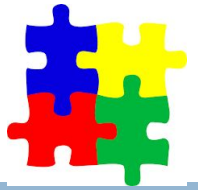
- Cannot simply compare whether two objects are equal by **==**
  - This is comparing their reference or name, which is unique
  - Try creating two identical Strings and testing whether they are equal
  - (We'll see later how to do it correctly)

- Impacts semantics of methods

References - Parameter Passing

- Parameters in a method can be passed either by:
  - Pass-by-value
    - Creates a copy of the parameter and passes that copy
    - Modifications in the method to the original element has no impact
  - Pass-by-reference
    - Directly passes a reference to the parameter
    - Modification to the original element are reflected

References - Parameter Passing



- Java is exclusively **passed-by-value**: primitive types and references to objects are **copied** to create method arguments
  - But (and this is where people get confused), a copy of a reference X is still pointing to **the same object X**. In contrast, a copy of integer i points to a different integer i.
  - Modifications to objects inside a method are reflected outside of the method, modifications to primitive types aren't.

- Other languages give you more flexibility to choose:
  - C++ allows you to specify methods in three ways:
    - swap(int x, int y) , swap(int* x, int* y), swap(int& x, int& y)

References - Swap Function

```
int a = 0;
Int b = 10;
swap(a,b);
System.out.println(a + " " + b);

void swap(int a, int b) {
        int tmp = a;
        a = b;
        b = tmp;
}
```

What will the different System.out.println() print if pass-by-value or pass-by-reference

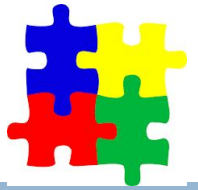
```
MyInt a = new MyInt(0);
MyInt b = new MyInt(10);
System.out.println(a);
System.out.println(b);
swap(a,b);
System.out.println(a);
System.out.println(b);
System.out.println(a.myInt);
System.out.println(b.myInt);

void swap(MyInt a, MyInt  b) {
        int tmp = a.myint;
        a.myint = b.myint;
        b.myint = tmp;
        MyInt tmpInt = a;
        a = b;
        b = tmpInt;
}
```

null

- Denotes the **absence** of a reference
  - Date date; or Date date = null;
- There is no equivalent for primitive types
  - Primitive types implicitly get initialised to default values
    - Try printing the value of int i and of Date date;
- Useful to explicitly state that no instance currently exists, but one may in the future.
- If a variable is **null**, cannot call a method or field on that object (doesn't exist)

static



- State/behaviour can sometimes be associated with a class rather than a specific instance of a class.
- A **static field** is created only once in the program's execution, despite being declared as part of a class
- A **static method** is invoked directly, without going through a specific instance
  - `Date.convertToString(pMonth)`
- What about the **main** method in Java?

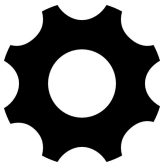
static

- **static** String dateUKFormat = “dd/mm/yyyy”
- **static** String dateUSFormat = “mm/dd/yyyy”
- **static** String convertToString(int month) {

```
        String monthSt = null;
        switch(month) {
            case 1: monthSt = “January”; break;
            case 2: monthSt =  “February”; break ;
            ...
        }
        return monthSt;
    }
```

This is a **switch** statement. Lookup the syntax!

When to use static



- Should method: **isDateEarlier** be static?
  - boolean isDateEarlierThan(Date date) {

    if (year < date.year) { return true; } ...

    }
  - Or **static** boolean isDateEarlierThan(Date date1, Date date2) {

    if (date1.year < date2.year) { return true;} ...

    }

- Good example of static methods: java.lang.Math
  - http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
  - Or find it by googling Java 8 Math

References in JavaHyperText

object

instance

class

static

null

field

method

pass-by-value

pass-by-reference