Introduction to Java

Handout-1d
Methods (i)

• *Method* is the OOP name for function
  – Must be declared always within a class

```java
optAccessQualifier returnType methodName ( optArgumentList ) optThrowsClause {
  ... // statements
}
```

Method “signature”

• You can several methods with the same name in the class
  – Same-name methods are said to *overload* the name
  – Methods with same name should do the same thing
Methods (ii)

• Primitive variables declared inside a method have undefined initial values

Class Fruit {
    int grams; // instance variable, will be initialized to 0 (zero)

    void someMethod() {
        int calories; // method variable, need to initialize before using
    }
}
Methods (iii)

• Calling a method in the same class

class Fruit {
    int grams;

    int totalCalories() {
        ... // Statements
    }

    void someMethod() {
        int i = totalCalories(); //same as i = this.totalCalories()
    }
}
Methods (v)

• Calling a method in a different class

class Cooking {
    int grams;

    Fruit apple = ...

    void otherMethod() {
        int i = apple.totalCalories(); // tell what object to use
    }
}
Methods (vi)

• Passing parameters to methods
  – Variables of primitive types are passed \textit{by value}
    • The argument’s value is copied and passed to the method. The method can change this copy, however this will not change the original argument
  – Objects are passed \textit{by reference}
    • The method is directly accessing the object. After returning from the method the object retains all changes made in the method
Methods (vii)

• Dynamic data structures

Ex:

class BinaryTree {
    private Object nodeData;
    private Tree left;  // left sub-tree
    private Tree right; // right sub-tree
    ...
}

Methods (viii)

• Per-instance and per-class members
  – `static` keyword makes something exist per-class, not per instance of that class

• There are four varieties of static
  – Data; the data belongs to the class, not individual instances of the class
  – Methods; these are methods that belong to the class
  – Blocks; these are blocks that are executed only once
  – Classes; these are classes that are nested in another class
Methods (ix)

• Static data

Ex:

class Employee {
    String name;    // per object field
    int salary;     // per-object field
    int employee_id; // per-object field

    static int total_employees; // per-class field (one only)

    ...
}

Methods – static data

• Inside the class static data is accessed by its name.
• Outside the class, static data can be accessed by:
  – Prefix it with the name of an object of that class OR
  – Prefix it with the name of the class

Ex:
Employee newhire = new Employee();

newhire.total_employees=1; // reference through the instance

Employee.total_employees=1; // reference through the class
Methods – static methods

• Static methods, aka class methods do class wide operations and do not apply to individual objects

Ex:

```java
class Employee {
    String name;
    int salary;
    int employee_id;
    static int employee_id;
    static int total_employees;
    static void clear() {
        total_employees = 0;
    }
}
```
Methods – static methods

- Access static method: better to call using the name of the class to avoid confusion with per-instance methods

Ex:

newhire.clear(); // reference through an instance

Employee.clear(); // better, reference through class
Methods – static method pitfalls

• Common error: reference per-object data from a static method

Ex:

```java
public static void main(String[] args) {
    salary = 50000;  // Compiler error
    Employee e = new Employee();
    e.salary = 50000; // This is ok
}
```
Methods – static blocks

• Block of code: statements within a pair of curly braces

• Static block is prefixed by `static`
  – Inside a class
  – Outside all methods
  – Most commonly used for initialization
  – Each static block is executed once only, when class if first loaded into the JVM
  – Can only access static data
Methods – static blocks

Ex:

```java
public class Employee {
    String name;
    int salary;
    int employee_id;
    static int total_employees;

    static {
        if (IncludingTempsAndContractors)
            total_employees = 100;
        else total_employees = 75;
    }
}
```
Methods – static classes

• Nested static class: the declaration of an entire class (methods, data fields, constructors) as a static member of another class
  – Nested purely for convenience
Modifiers - final

- Makes something constant. Can be applied to code and data
  - When reference variable is declared final you can’t make that variable point at some other object
  - The reference is final not the referenced object
Modifiers - final

Ex:

```java
final static int myTotal = 100; // constant data
final Fruit banana = new Fruit(100, 30); // constant reference
Fruit lemon = new Fruit();
banana = lemon; // compilation error
    // cannot assign a value to final variable banana
```
Modifiers – final

• Method arguments can be marked as final

Ex:

```java
void someMethod(final MyClass c) {
    c.field = 7; // Ok
    c = new MyClass(); // compilation error
}
```
Modifiers - `final`

- *Blank final variable*: a final variable of any kind that does not have an initializer
  - Must be assigned a value; that value can be assigned only once
  - If you give a value to a blank final in a constructor, then you must assign it a value in each constructor
Modifiers - final

Ex:

Class Fruit {
    final String consumer; // blank final variable

    Fruit (String s) {   // constructor
        consumer = s; // the blank final is now initialized
    }
    ... // more stuff
}

Access modifiers

- **private**: members are not accessible outside the class
- **None** (aka “package access”): members are accessible from classes in the same package
- **protected**: members are accessible from the package AND in subclasses of this class
  - This is *less* protected than the default of package access
- **public**: members are accessible anywhere the class is accessible
Access modifiers - private

- Making a constructor private prevents the class from being instantiated by other classes
- Making a method private means it can only be called by another method in the same class

Ex:

class Fruit {
    private int grams;
    private int caloriesPerGram;

    private Fruit() {    // constructor
        grams = 0;
        caloriesPerGram = 0;
    }
}
Access modifiers - package

Ex:

class Employee { // package access
    String name;
    int salary;
    static int total_employees;

    static void clear() {
        total_employees = 0;
    }
}

Access modifiers - protected

Ex:

class Employee {  // package access
    protected String name;
    protected int salary;
    static int total_employees;

    protected void giveRaise(int amount) {
        salary = salary + amount;
    }
}
Access modifiers - public

Ex:

```java
public class Employee {  // public access
    public static void main() {
        ...
        ...
    }
}
```