PRIMER JAVA

LECTURER: Ms.Tran Le Nhu Quynh
Email (for submit student's homework) :
quynhtranlenhu@gmail.com
Email: nquynh@hcmuaf.edu.vn
Version : 2021- 2022

CONTENTS

1. INTRODUCE JAVA
2. OBJECT AND CLASS
3. OPERATORS
4. JAVA PROGRAMING RULES
5. DATA TYPES AND IT'S OPERATORS
6. TEST – JUNIT TEST
7. EXECERCISES

INTRODUCE JAVA

WHAT'S JAVA?.....

HISTORY OF JAVA

- Java was developed by a team led by James Gosling at Sun Microsystems.
- Originally called Oak, it was designed in 1991 for use in embedded chips in consumer electronic appliances.
- In 1995, renamed Java, it was redesigned for developing Internet applications

The Java Language Specification, API, JDK, IDE

- Computer languages have strict rules of usage. You need to follow the rules when writing a program, then the computer can understand it.
- **The Java language specification** and **Java API** define the Java standard.
  - **The Java language specification** is a technical definition of the language that includes the syntax and semantics of the Java programming language.
  - The **application program interface** (API) contains predefined classes and interfaces for developing Java programs.
- **The Java language specification** is stable, but the **API** is still expanding.

The Java Language Specification, API, JDK, IDE

- Java is a full-fledged and powerful language that can be used in many ways.
  - **Java Standard Edition** (Java SE): to develop client-side standalone applications or applets.
  - **Java Enterprise Edition** (Java EE): to develop server-side applications, such as Java servlets and Java Server Pages.
  - **Java Micro Edition** (Java ME): to develop applications for mobile devices, such as cell phones.

The Java Language Specification, API, JDK, IDE

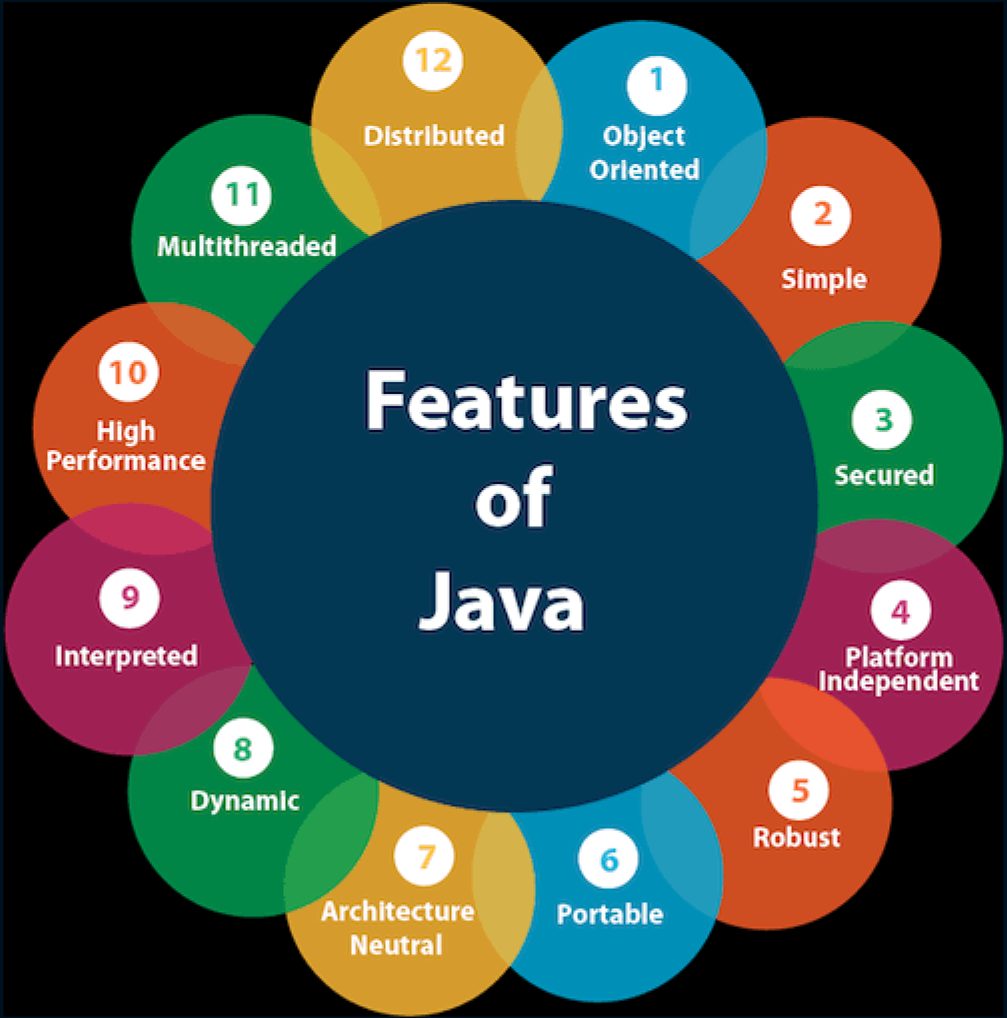
- Use Java SE to introduce Java programming in this subject.
- There are many versions of Java SE. Sun releases each version with a Java Development Toolkit (JDK).
- For Java SE 6, the Java Development Toolkit is called JDK 1.6 (also known as Java 6 or JDK 6).

The Java Language Specification, API, JDK, IDE

- Use a Java development tool (e.g., NetBeans, Eclipse) - software that provides an **integrated development environment** (**IDE**) for rapidly developing Java programs.
    - Editing, compiling, building, debugging, and online help are integrated in one graphical user interface.

“

- WHAT IS JAVA ‘S FEATURES ?.....


Features of Java
12 Distributed
1 Object Oriented
2 Simple
3 Secured
4 Platform Independent
5 Robust
6 Portable
7 Architecture Neutral
8 Dynamic
9 Interpreted
10 High Performance
11 Multithreaded

OBJECT-ORIENTED

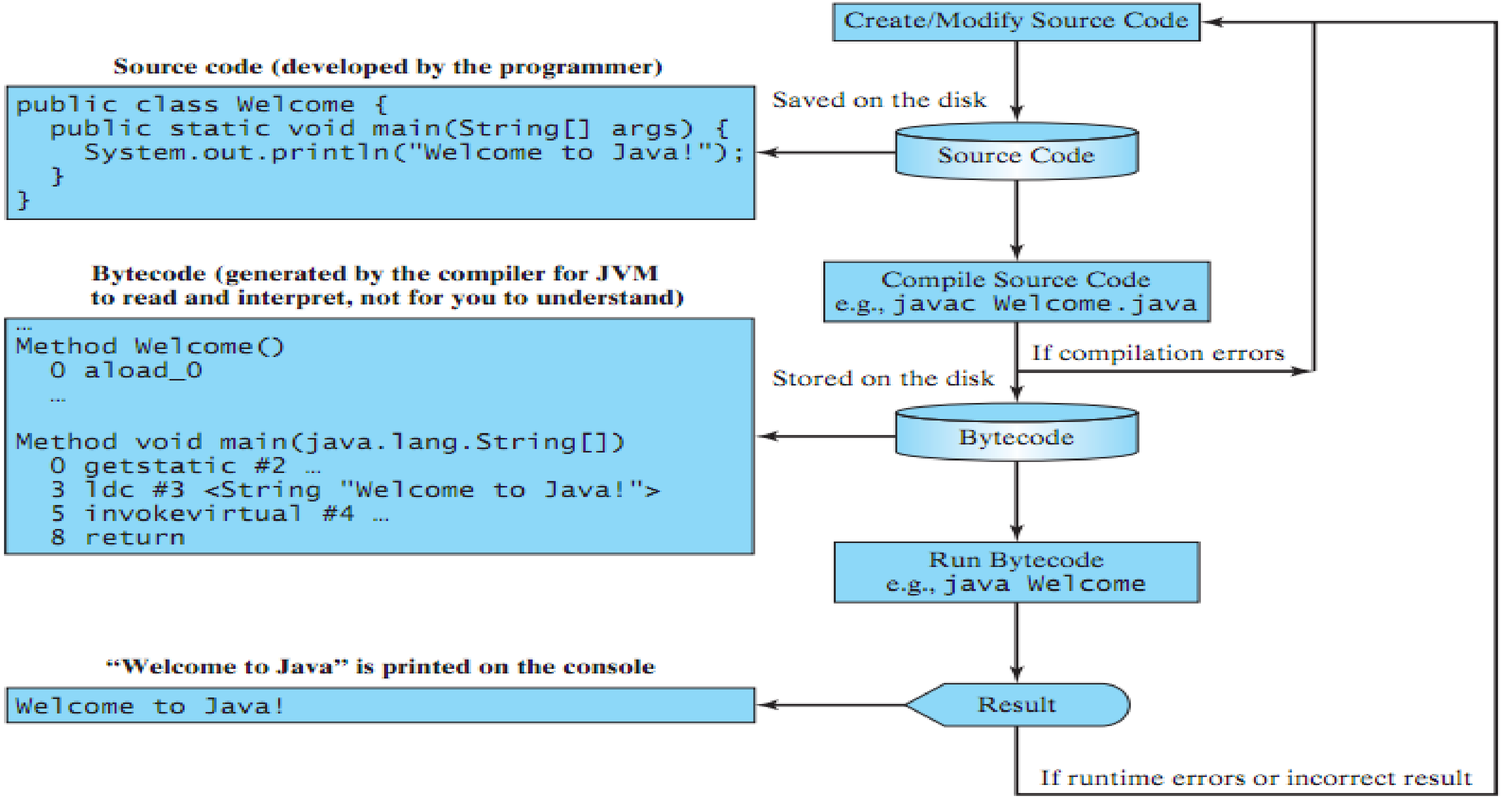
- Java is an **object-oriented** programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

HOW IS JAVA WORK ?.....

Creating, Compiling, and Executing



Create/Modify Source Code

Saved on the disk

Source Code

Source code (developed by the programmer)

```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Compile Source Code
e.g., javac Welcome.java

If compilation errors

Stored on the disk

Bytecode

Bytecode (generated by the compiler for JVM to read and interpret, not for you to understand)

```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

Run Bytecode
e.g., java Welcome

Result

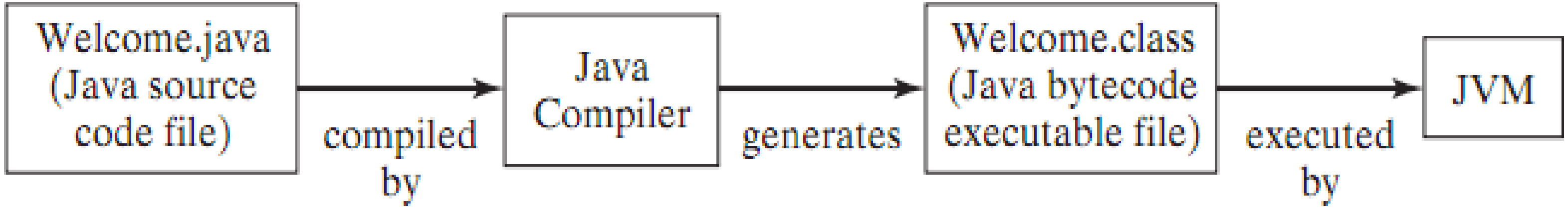
If runtime errors or incorrect result

"Welcome to Java" is printed on the console

```
Welcome to Java!
```

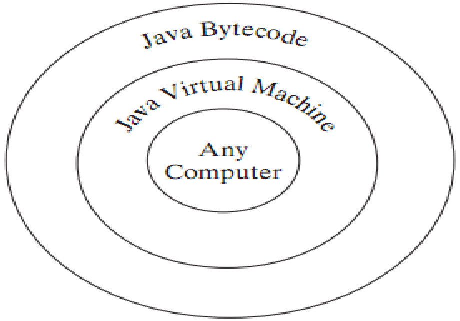
Creating, Compiling, and Executing

- If there are no syntax errors, the compiler generates a bytecode file with a *.class* extension.
- The Java language is a high-level language while Java bytecode is a low-level language.



Creating, Compiling, and Executing

- The bytecode is similar to machine instructions and can run on any platform that has a **Java Virtual Machine** (**JVM**).
- The virtual machine is a program that interprets Java bytecode.
- Java bytecode can run on a variety of hardware platforms and operating systems.



“

- ***WHAT IS JAVA USE FOR ?.....***

Mobile Phones

If you have an Android phone you use Java every day! Android apps – and indeed the Android operating system! - are written in Java, with Google's API, which is similar to JDK.

Point of Sale Systems

Java is also used in the creation of PoS systems, helping businesses exchange goods or services for money from their customers.

Video Games

One of the most popular games of all time, Minecraft, was written in Java by Mojang. Minecraft is a sandbox construction game, where you can build anything you can imagine.

Trading Applications

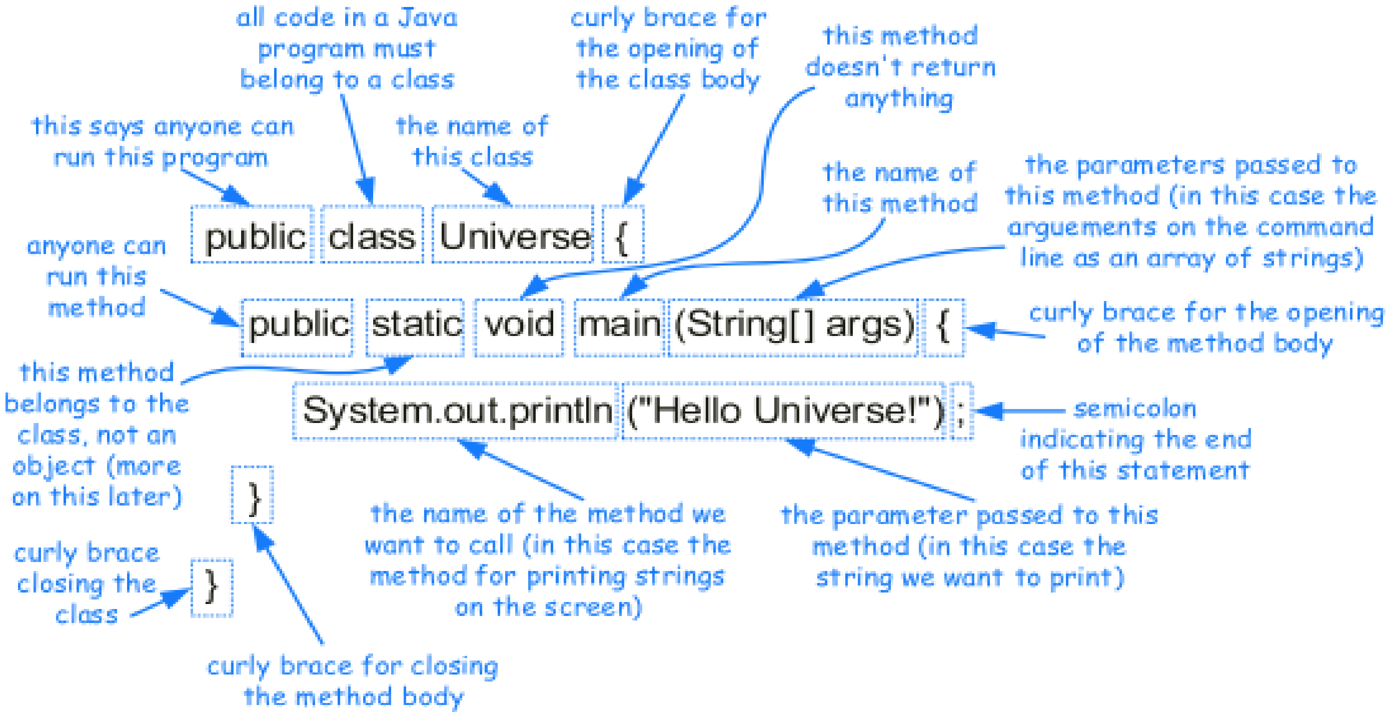
Several third-party trading applications use Java. Murex, which is used by many banks for front to back connectivity, is also written in Java.

Big Data Technologies

The Java platform is very popular in writing high-performance systems. Hadoop and ElasticSearch are both written in Java and are often used in Big Data projects.

EXAMPLE JAVA CODE

A SIMPLE JAVA PROGRAM



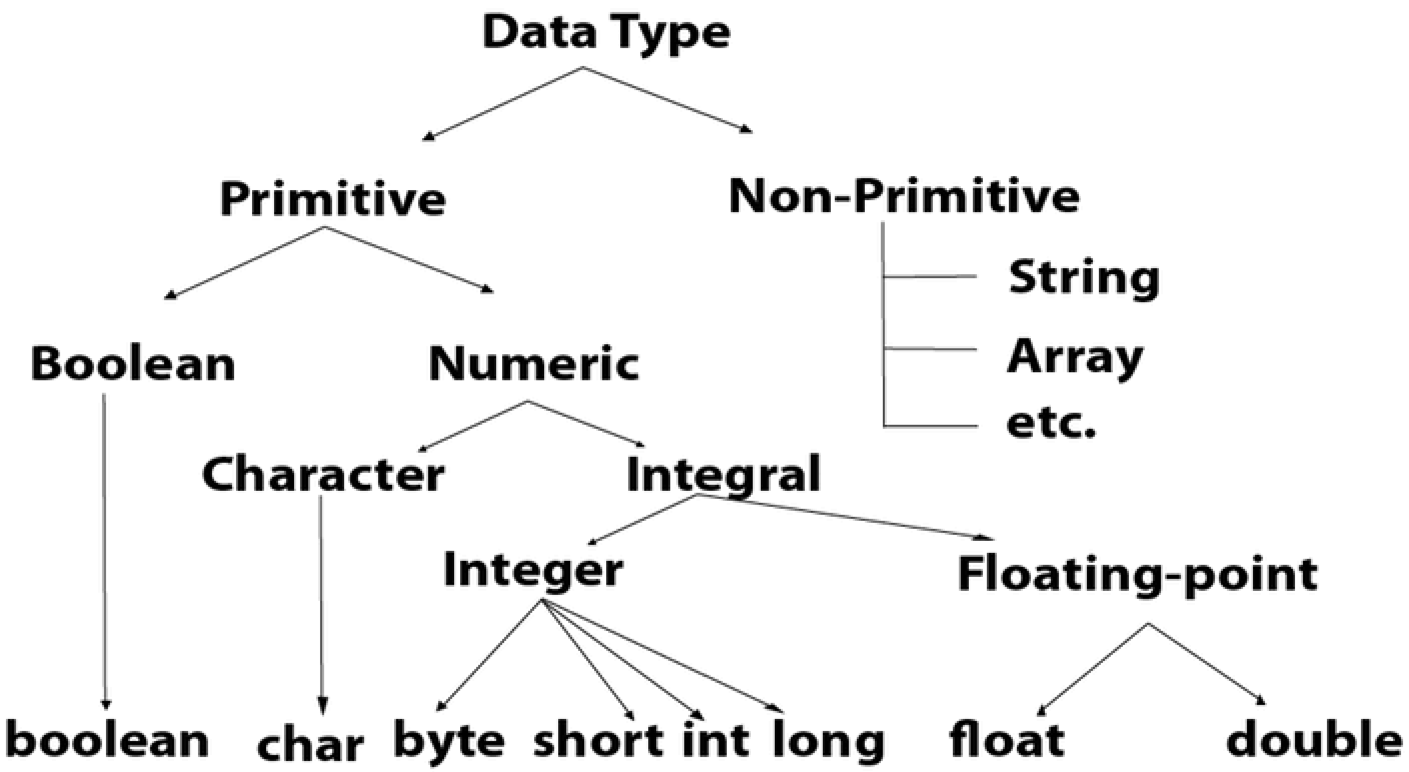
```
public class Universe {
    public static void main(String[] args) {
        System.out.println("Hello Universe!");
    }
}
```

all code in a Java program must belong to a class
curly brace for the opening of the class body
this method doesn't return anything
this says anyone can run this program
the name of this class
the name of this method
the parameters passed to this method (in this case the arguements on the command line as an array of strings)
anyone can run this method
curly brace for the opening of the method body
this method belongs to the class, not an object (more on this later)
semicolon indicating the end of this statement
the name of the method we want to call (in this case the method for printing strings on the screen)
the parameter passed to this method (in this case the string we want to print)
curly brace closing the class
curly brace for closing the method body

Figure 1.1: A "Hello Universe!" program.

BASIC JAVA PROGRAMING

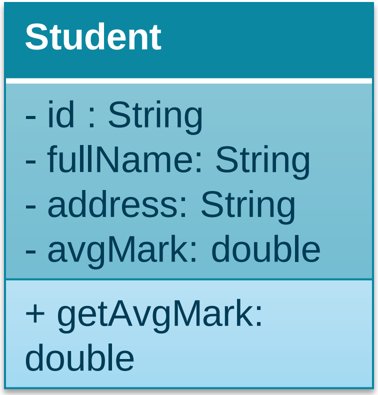
DATA TYPES IN JAVA



OBJECT VS CLASS

- REAL LIFE

- CLASS

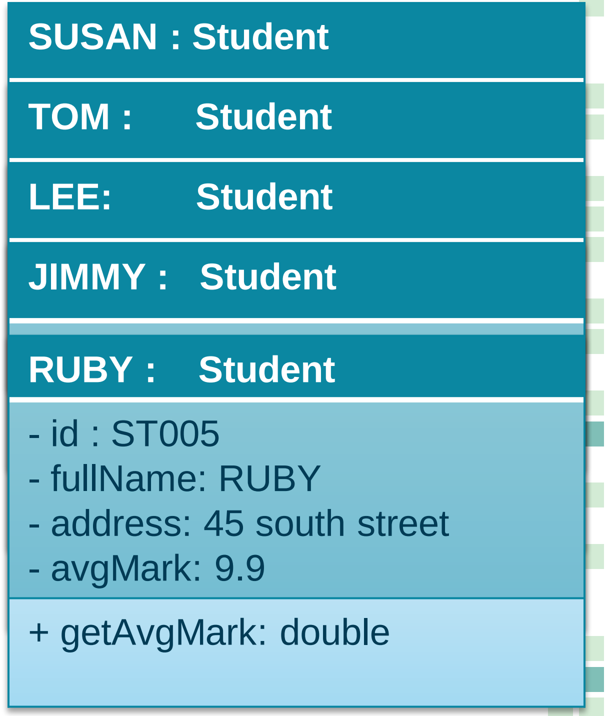


Student

- id : String
- fullName: String
- address: String
- avgMark: double

+ getAvgMark: double

- OBJECT



SUSAN : Student

TOM : Student

LEE: Student

JIMMY : Student

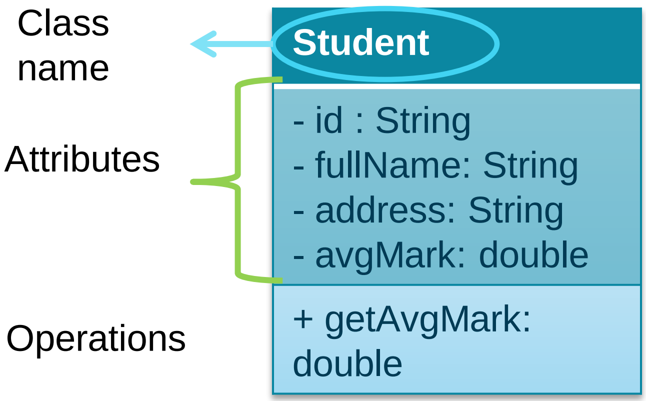
RUBY : Student

- id : ST005
- fullName: RUBY
- address: 45 south street
- avgMark: 9.9

+ getAvgMark: double

CLASS VS OBJECT IN UML

- CLASS



Class name

Attributes

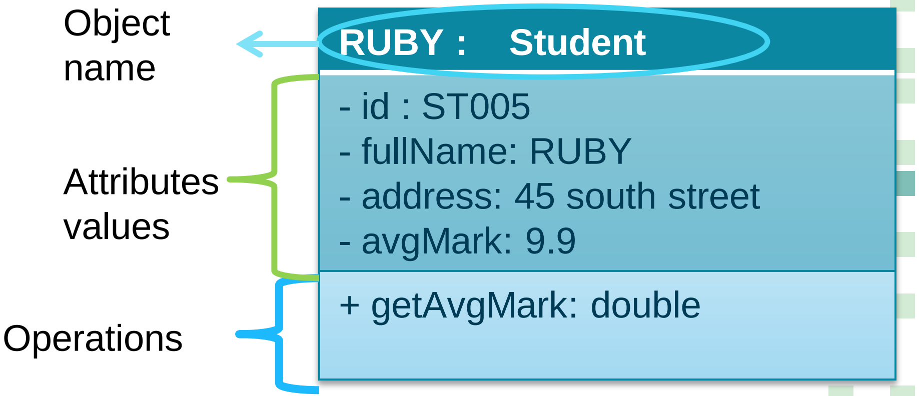
Operations

Student

- id : String
- fullName: String
- address: String
- avgMark: double

+ getAvgMark: double

- OBJECT is instance of CLASS



Object name

Attributes values

Operations

RUBY : Student

- id : ST005
- fullName: RUBY
- address: 45 south street
- avgMark: 9.9

+ getAvgMark: double

CLASS STRUCTURE

optional-package-declaration

optional-imports

public class ClassName_{

attributes *(**optional**-variable-declarations)*

constructor

operations

test

}

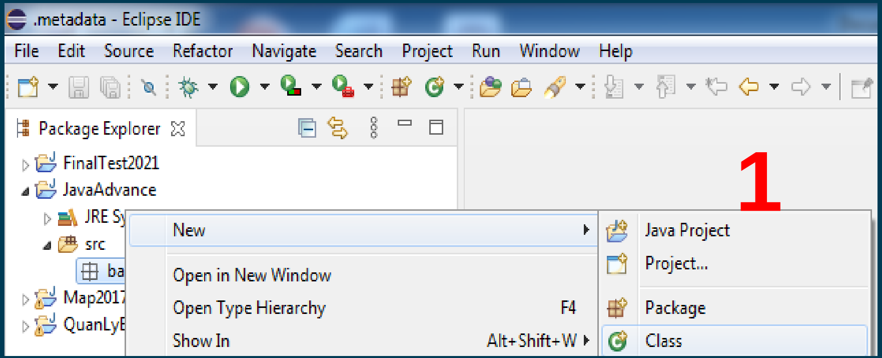
Create class

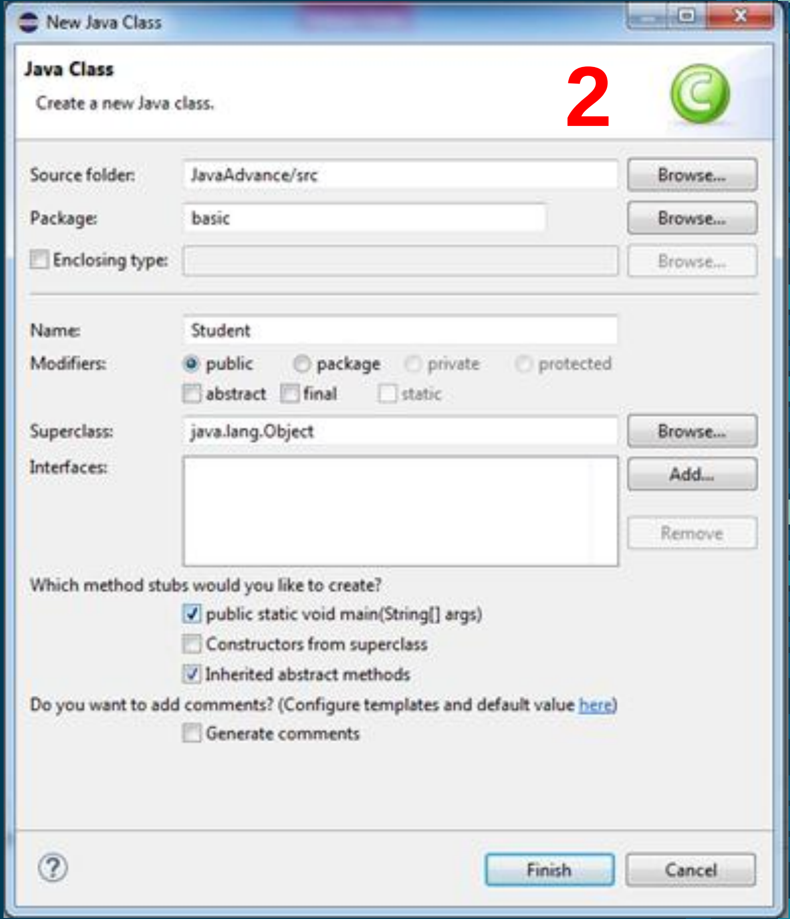
Class name rule

- Start with a capital letter
- Be centered in the top compartment
- Not begin by number
- **Example**: Student => good class name
  SinhVien => good class name
  sinhvien => bad class name

Exercise 1

Step 0: create Student class





Java Class

Create a new Java class.

Source folder: JavaAdvance/src

Package: basic

□ Enclosing type:

Name: Student

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

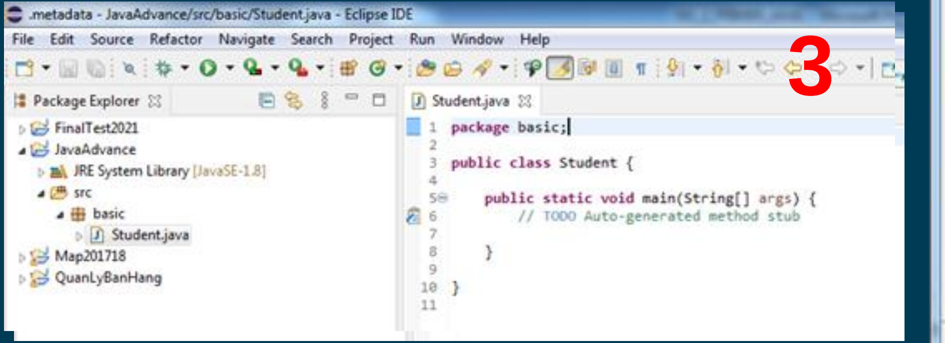
☑ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments


.metadata - JavaAdvance/src/basic/Student.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
3
Package Explorer
FinalTest2021
JavaAdvance
JRE System Library [JavaSE-1.8]
src
basic
Student.java
Map201718
QuanLyBanHang
Student.java

```
package basic;

public class Student {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}

```

Create attributes

(optional)

CLASS STRUCTURE

optional-package-declaration

optional-imports

public class ClassName {

attributes *(**optional**-variable-declarations)*

constructor

operations (methods)

test

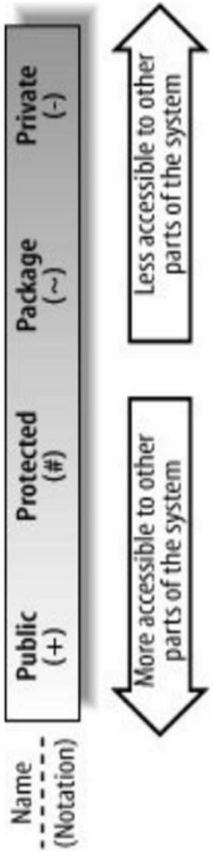
}

Attribute

- Syntax:
- Visibility + data type + name of attribute

VISIBILITY *+ data type + name of attribute*

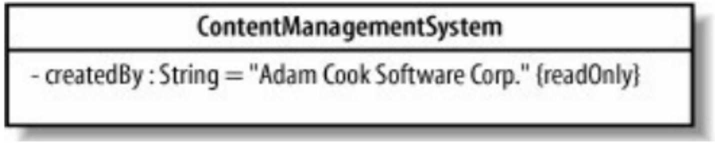
Visibility

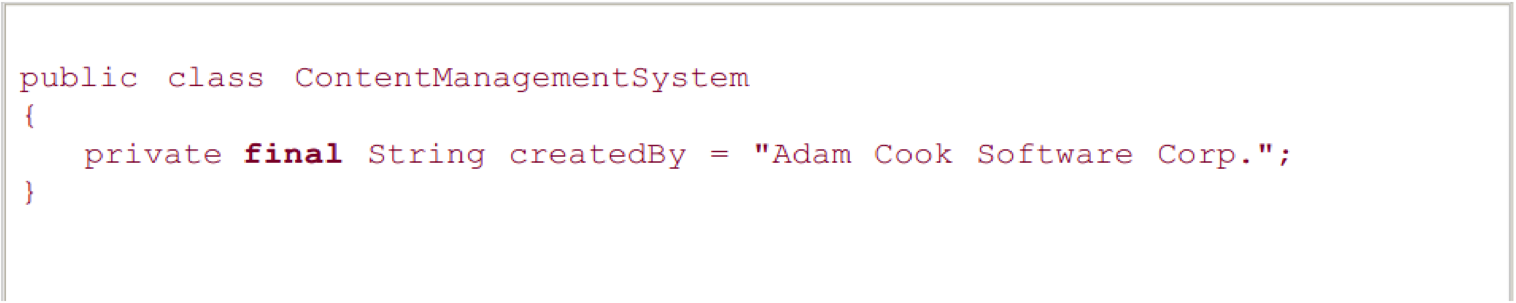

Private
(-)
Package
(~)
Protected
(#)
Public
(+)
Name
(Notation)
Less accessible to other parts of the system
More accessible to other parts of the system

Modifier	Visibility outside the class
private	None
No modifier (default)	Classes in the package
protected	Classes in package and subclasses inside or outside the package
public	All classes

Read Only property

Figure 4-13. The createdBy attribute in the ContentManagementSystem class is given a default initial value and a property of readOnly so that the attribute cannot be changed throughout the lifetime of the system


ContentManagementSystem
- createdBy : String = "Adam Cook Software Corp." {readOnly}



```
public  class  ContentManagementSystem
{
    private final String createdBy = "Adam Cook Software Corp.";
}
```

> *Visibility + **data type + name of attribute***

VARIABLE

Variable in Java

- Local variable
- Instance variable
- Static variable

Local variable

- A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.

Instance variable

- A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
- It is called instance variable because its value is instance specific and is not shared among instances.

Static variable

- A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example:

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Initializing variables

- **Variables** often have initial values.
- Declare a **variable** and initialize it in one step: **int** count = 1;
- The next two statements are same: **int** count; count = 1;
- You can also use a shorthand form to declare and initialize variables of the same type together.

  int i = 1, j = 2;
- TIP:
  - A **variable** declared in a method must be assigned a value before it can be used.
  - You should declare a **variable** and assign its initial value in one step ➔ make the program easy to read and avoid programming errors.

Assignment Statements

- You can assign a **value** to a variable by using an *assignment statement*.

```
variable = expression;
```

Assignment Expressions

- An **expression** represents a computation involving *values*, *variables*, and *operators* that, taking them together, evaluates to a value.

```
int x = 1;
double radius = 1.0;
x = 5 * (3 / 2) + 3 * 2;
x = y + 1;
area = radius * radius * 3.14159;
```

- To assign a value to a **variable**, the *variable name* must be on the *left* of the *assignment operator*:

  1 = x ➔ Right or wrong?

Named Constants

- The value of a **variable** may *change* during the execution of a program, but a named constant or simply constant represents permanent data that never changes.
  - In **ComputeArea** program, π is a **constant**. If you use it frequently, you don't want to keep typing 3.14159 ➔ declare a constant for π
- **Syntax**:

  final datatype CONSTANT_NAME = VALUE;
- By convention, constants are named in **uppercase**: **PI**, not **pi** or **Pi**.

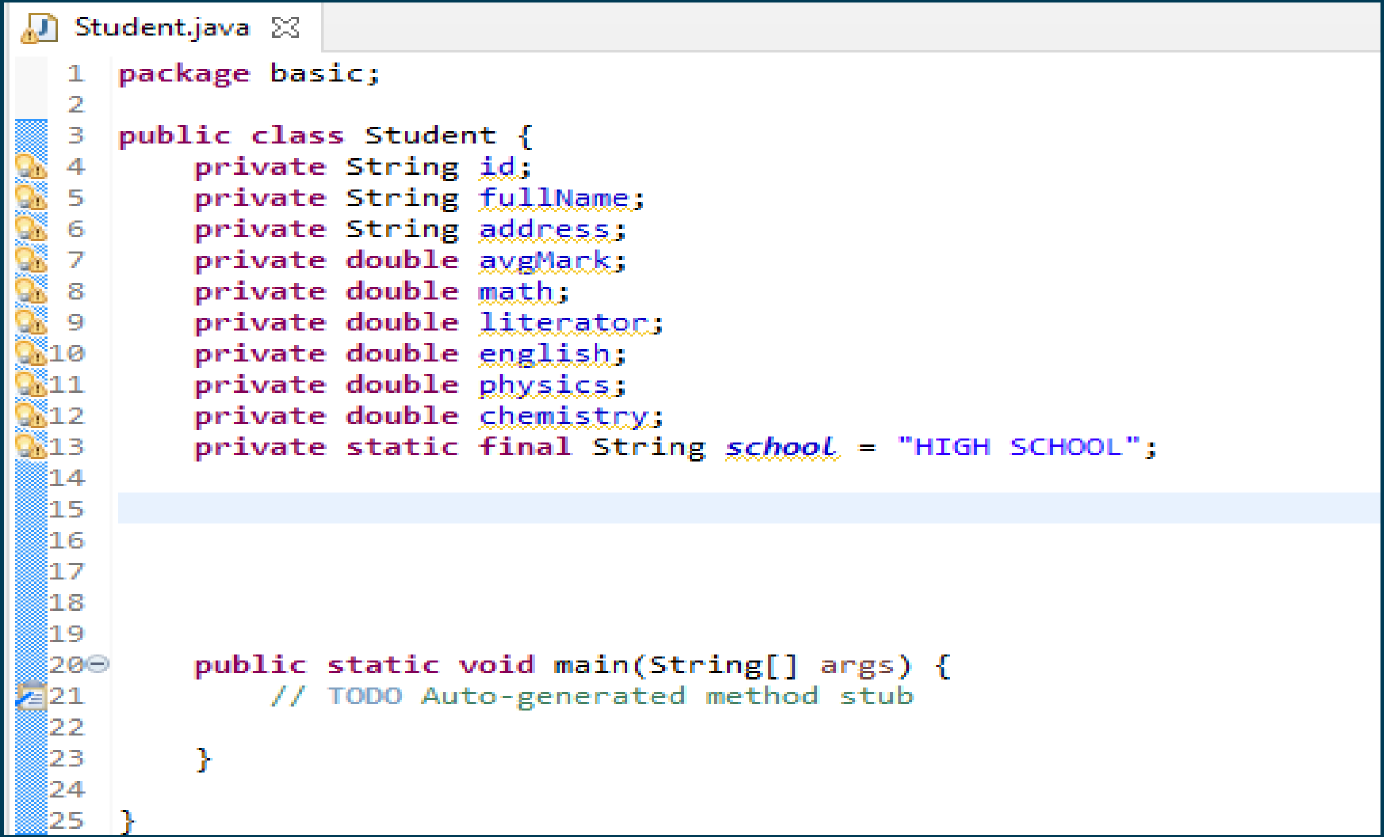
Named Constants

- There are three benefits of using constants:
  - (1) you don’t have to repeatedly type the same value;
  - (2) if you have to change the constant value (e.g., from 3.14 to 3.14159 for PI), you need to change it only in a single location in the source code;
  - (3) a descriptive name for a constant makes the program easy to read.

Exercise 1

Step1: create attributes of Student class:

- Id : String
- Full name : String
- Address : String
- Average Mark: double
- Math: double
- Literature: double
- English: double
- Physics: double
- Chemistry: double
- School : String // never change value and value is “ HIGH SCHOOL”



```java
package basic;

public class Student {
	private String id;
	private String fullName;
	private String address;
	private double avgMark;
	private double math;
	private double literator;
	private double english;
	private double physics;
	private double chemistry;
	private static final String school = "HIGH SCHOOL";






	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

Constructor

(optional)

Constructor

- A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.
- A class may be has many constructors
- Constructor hasn't parameter as default (*don't need to create*)
- Constructor has parameters (*must create*)

Constructor has parameters

```
visibility  datatype attributeName1;
visibility  datatype attributeName2;
....
public className (dataType1 parameterName1,
                                    dataType 2 parameterName2,....){
 this. attributeName1 =  parameterName1;
 this. attributeName2 =  parameterName2;
 ...
 }
```

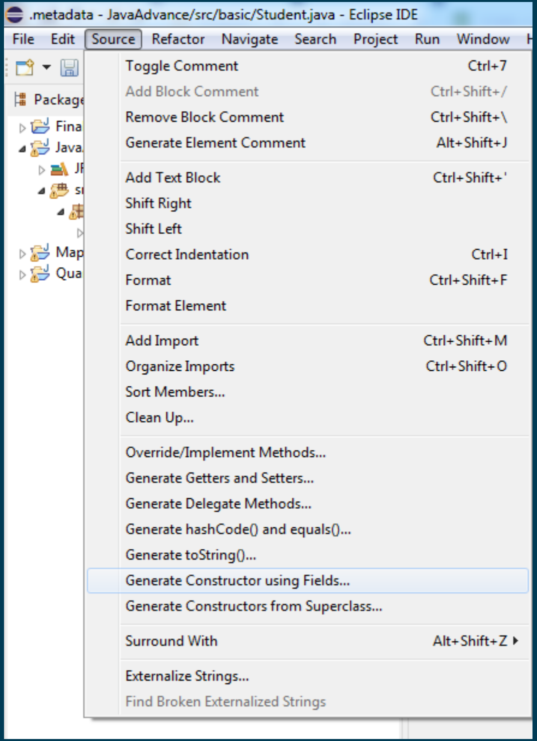
Exercise 1

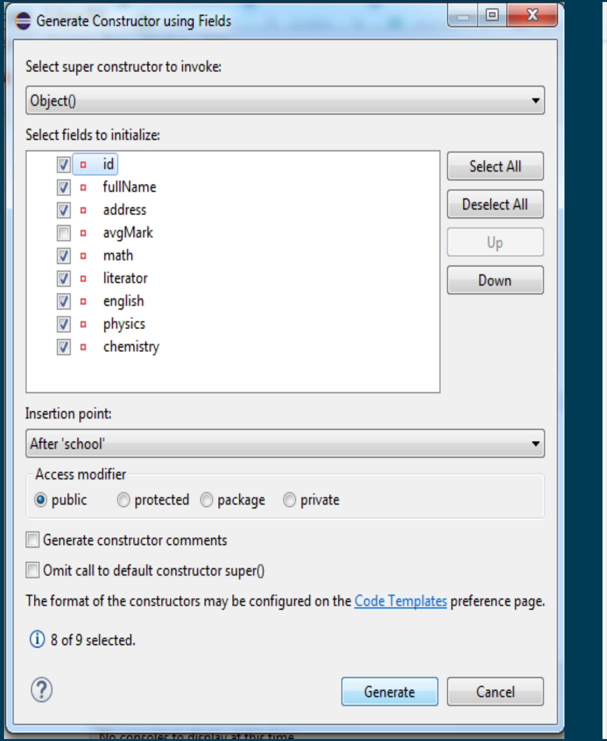
Step 2: create constructor of Student class:

- Constructor default
- Constructor has parameters : id , full name , address, average, mark, math, literature, english, physics, chemistry.
- Constructor has parameters: id, full name. Rest of attribute will be update values by set() method

Two ways to create Constructor:

1) *Coding by typing*
2) *Coding by auto generate code of Eclipse*


.metadata - JavaAdvance/src/basic/Student.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window
Package
Toggle Comment Ctrl+7
Add Block Comment Ctrl+Shift+/
Remove Block Comment Ctrl+Shift+\
Generate Element Comment Alt+Shift+J
Add Text Block Ctrl+Shift+'
Shift Right
Shift Left
Correct Indentation Ctrl+I
Format Ctrl+Shift+F
Format Element
Add Import Ctrl+Shift+M
Organize Imports Ctrl+Shift+O
Sort Members...
Clean Up...
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Generate hashCode() and equals()...
Generate toString()...
Generate Constructor using Fields...
Generate Constructors from Superclass...
Surround With Alt+Shift+Z
Externalize Strings...
Find Broken Externalized Strings



Select super constructor to invoke: Object()

Select fields to initialize:

- ☑ id
- ☑ fullName
- ☑ address
- ☐ avgMark
- ☑ math
- ☑ literator
- ☑ english
- ☑ physics
- ☑ chemistry

Insertion point: After 'school'

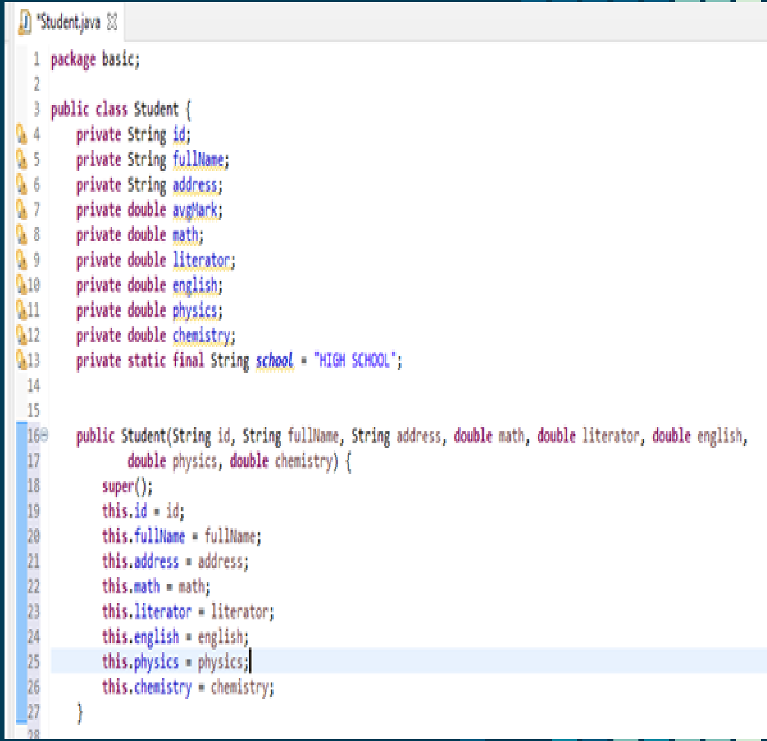
Access modifier: ● public ○ protected ○ package ○ private

☐ Generate constructor comments

☐ Omit call to default constructor super()

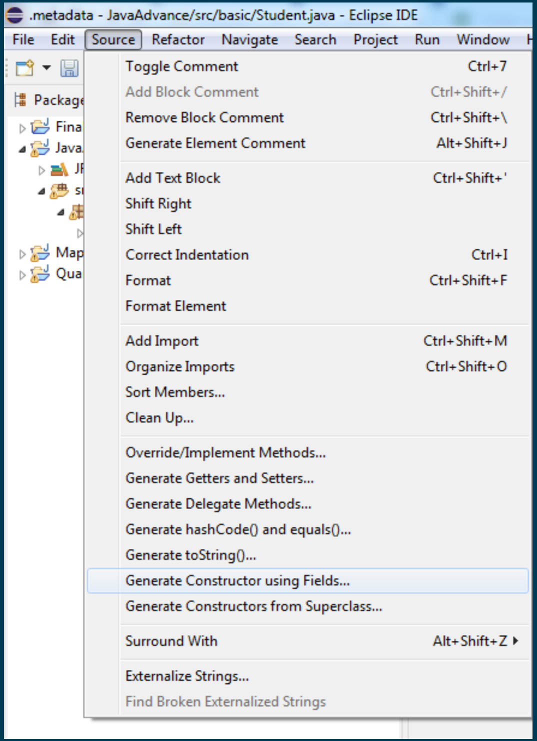
The format of the constructors may be configured on the Code Templates preference page.

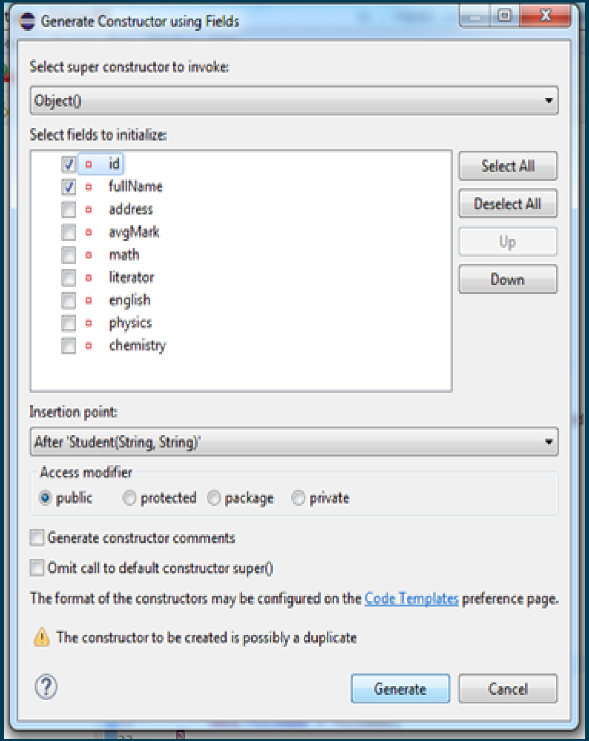
8 of 9 selected.



```java
package basic;

public class Student {
    private String id;
    private String fullName;
    private String address;
    private double avgMark;
    private double math;
    private double literator;
    private double english;
    private double physics;
    private double chemistry;
    private static final String school = "HIGH SCHOOL";


    public Student(String id, String fullName, String address, double math, double literator, double english,
            double physics, double chemistry) {
        super();
        this.id = id;
        this.fullName = fullName;
        this.address = address;
        this.math = math;
        this.literator = literator;
        this.english = english;
        this.physics = physics;
        this.chemistry = chemistry;
    }
```

A class can has many constructors.
Now, we want to initial Student class just has 2 paramters (id and fullName). Rest of parameters will be update later by set() method


.metadata - JavaAdvance/src/basic/Student.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Package
Fina
Java
JF
sı
Map
Qua
Toggle Comment Ctrl+7
Add Block Comment Ctrl+Shift+/
Remove Block Comment Ctrl+Shift+\
Generate Element Comment Alt+Shift+J
Add Text Block Ctrl+Shift+'
Shift Right
Shift Left
Correct Indentation Ctrl+I
Format Ctrl+Shift+F
Format Element
Add Import Ctrl+Shift+M
Organize Imports Ctrl+Shift+O
Sort Members...
Clean Up...
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Generate hashCode() and equals()...
Generate toString()...
Generate Constructor using Fields...
Generate Constructors from Superclass...
Surround With Alt+Shift+Z
Externalize Strings...
Find Broken Externalized Strings



Select super constructor to invoke:
Object()

Select fields to initialize:

- ☑ id
- ☑ fullName
- □ address
- □ avgMark
- □ math
- □ literator
- □ english
- □ physics
- □ chemistry

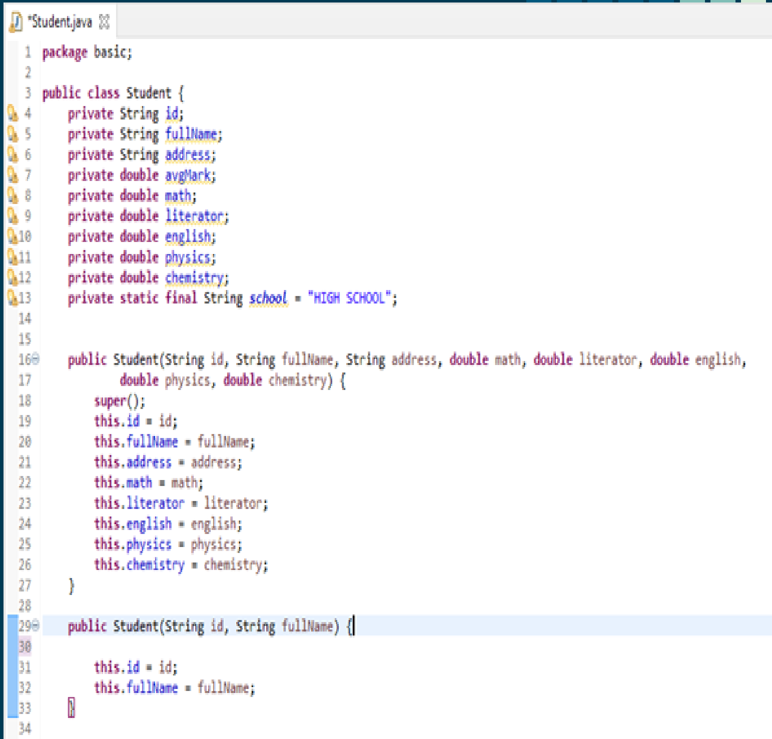
Insertion point:
After 'Student(String, String)'

Access modifier
● public ○ protected ○ package ○ private

□ Generate constructor comments
□ Omit call to default constructor super()

The format of the constructors may be configured on the Code Templates preference page.

⚠ The constructor to be created is possibly a duplicate



```
package basic;

public class Student {
    private String id;
    private String fullName;
    private String address;
    private double avgMark;
    private double math;
    private double literator;
    private double english;
    private double physics;
    private double chemistry;
    private static final String school = "HIGH SCHOOL";


    public Student(String id, String fullName, String address, double math, double literator, double english,
            double physics, double chemistry) {
        super();
        this.id = id;
        this.fullName = fullName;
        this.address = address;
        this.math = math;
        this.literator = literator;
        this.english = english;
        this.physics = physics;
        this.chemistry = chemistry;
    }

    public Student(String id, String fullName) {

        this.id = id;
        this.fullName = fullName;
    }

```

OPERATIONS

(METHODS)

Operator

- Syntax operator non parameter:

```
visibility returnType operatorName( ){
  //TODO
}
```

- Syntax operator with parameters:

```
visibility returnType operatorName(dataType1 parameterName1,
                                   dataType 2 parameterName2,…){
  //TODO

}
```

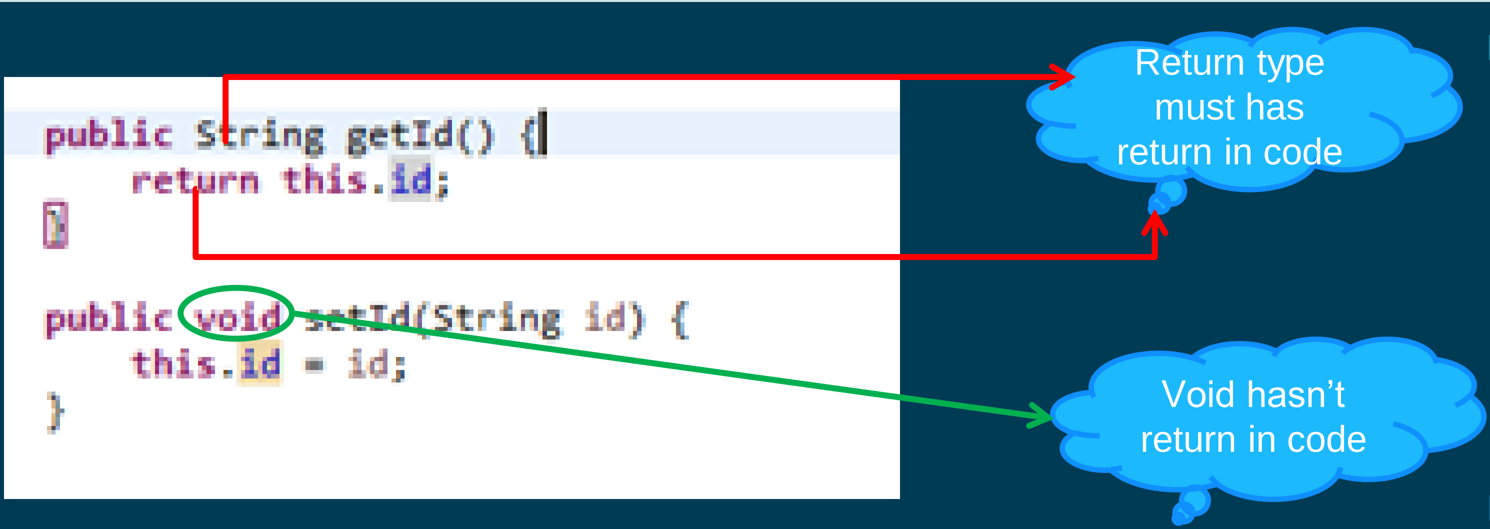
Return type

- A **return statement** causes the program control to transfer back to the caller of a method. Every method in Java is declared with a return type and it is mandatory for all java methods.
- A return type may be:
  - a **primitive type** like **int, float, double,…**
  - a **reference type** (**class types, array types, interface types..**)
  - **void type**(returns nothing => Not using **return** in code).

Exercise 1

Step 3: create get(),set() of Student class:

How to write get(), set() method by manual

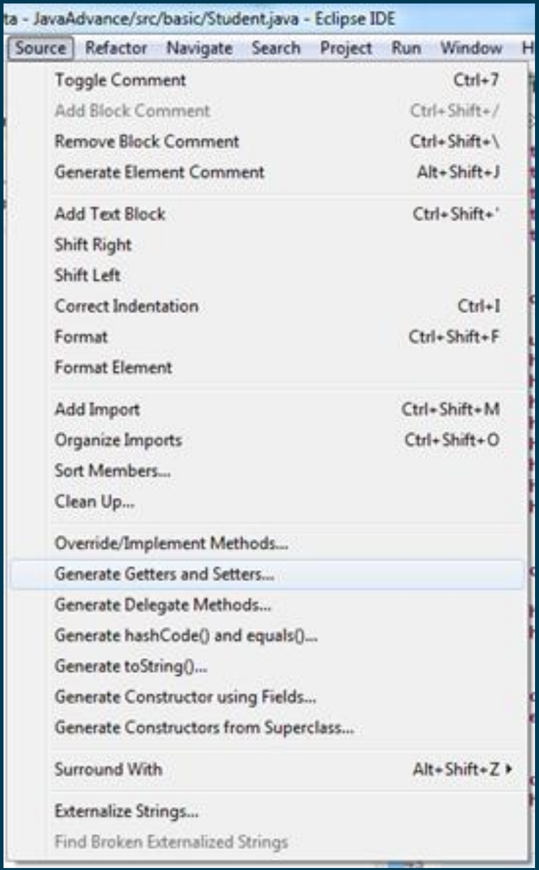


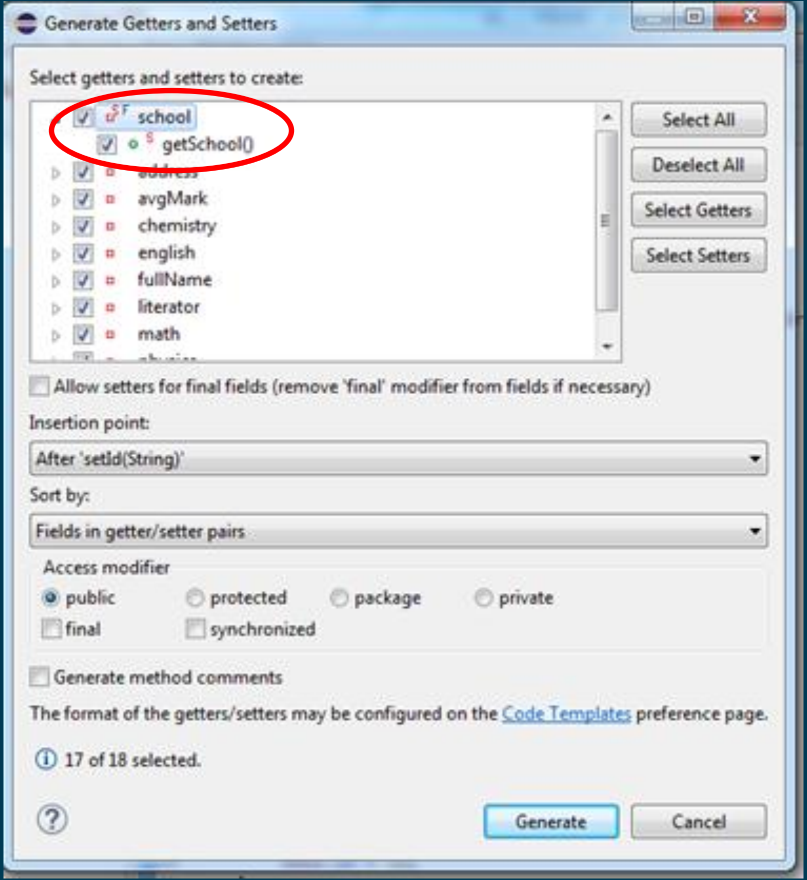
```
public String getId() {
    return this.id;
}

public void setId(String id) {
    this.id = id;
}
```

Return type must has return in code

Void hasn't return in code

How to write get(), set() method by code generator


ta - JavaAdvance/src/basic/Student.java - Eclipse IDE
Source Refactor Navigate Search Project Run Window H
Toggle Comment Ctrl+7
Add Block Comment Ctrl+Shift+/
Remove Block Comment Ctrl+Shift+\
Generate Element Comment Alt+Shift+J
Add Text Block Ctrl+Shift+'
Shift Right
Shift Left
Correct Indentation Ctrl+I
Format Ctrl+Shift+F
Format Element
Add Import Ctrl+Shift+M
Organize Imports Ctrl+Shift+O
Sort Members...
Clean Up...
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Generate hashCode() and equals()...
Generate toString()...
Generate Constructor using Fields...
Generate Constructors from Superclass...
Surround With Alt+Shift+Z
Externalize Strings...
Find Broken Externalized Strings



Select getters and setters to create:

- ☑ school
  - ☑ getSchool()
- ☑ address
- ☑ avgMark
- ☑ chemistry
- ☑ english
- ☑ fullName
- ☑ literator
- ☑ math
- ☑ physics

□ Allow setters for final fields (remove 'final' modifier from fields if necessary)

Insertion point: After 'setId(String)'

Sort by: Fields in getter/setter pairs

Access modifier

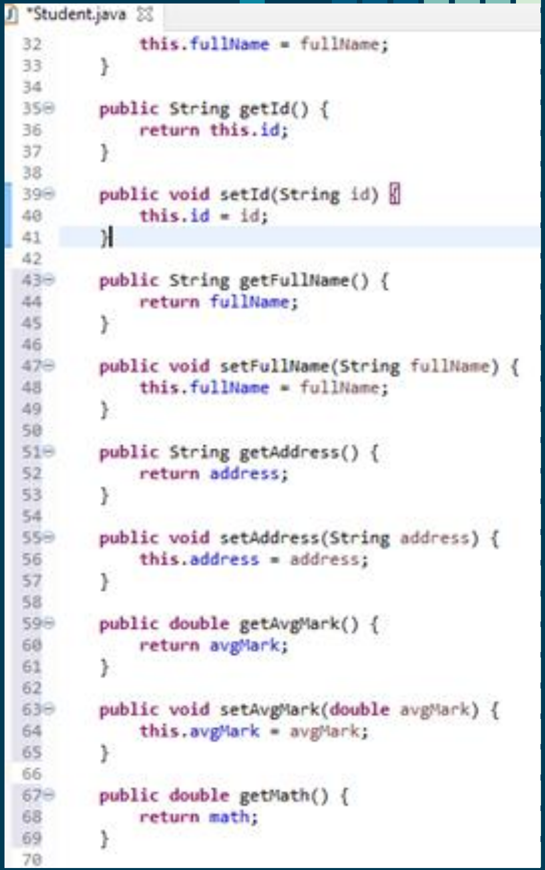
● public ○ protected ○ package ○ private

□ final □ synchronized

□ Generate method comments

The format of the getters/setters may be configured on the Code Templates preference page.

ⓘ 17 of 18 selected.



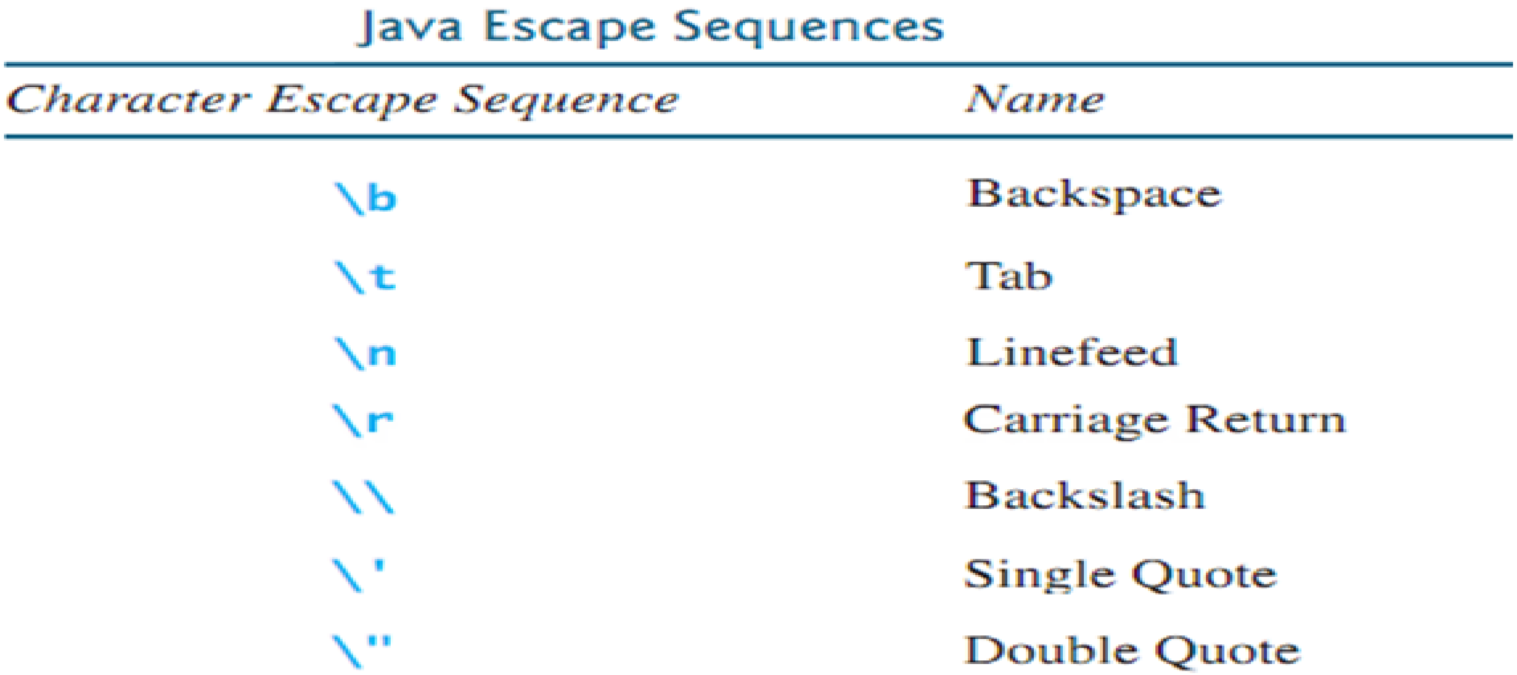
```
        this.fullName = fullName;
    }

    public String getId() {
        return this.id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public double getAvgMark() {
        return avgMark;
    }

    public void setAvgMark(double avgMark) {
        this.avgMark = avgMark;
    }

    public double getMath() {
        return math;
    }

```

Exercise 1

Step 3: create toString() of Student class:

- *The toString() method returns the string representation of the object.*
- *If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation*.

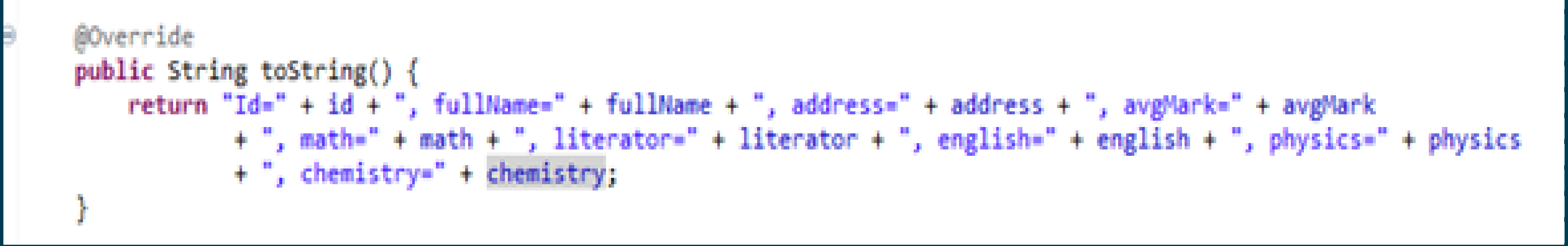
Escape Sequences for Special Characters



Java Escape Sequences

Character Escape Sequence	*Name*
`\b`	Backspace
`\t`	Tab
`\n`	Linefeed
`\r`	Carriage Return
`\\`	Backslash
`\'`	Single Quote
`\"`	Double Quote

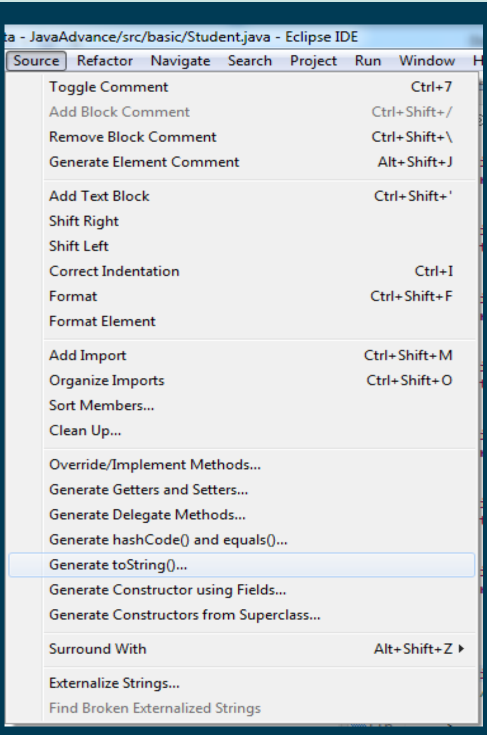
Creating toString() method by manual

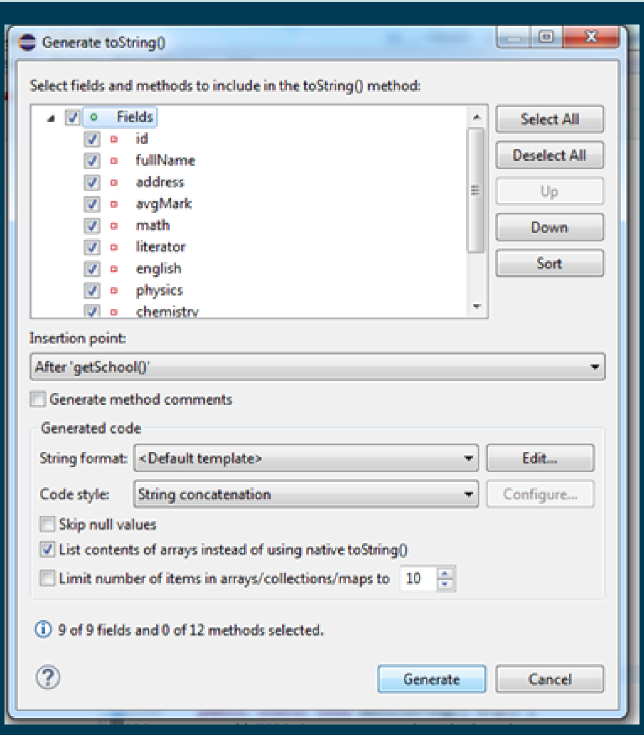


```
@Override
public String toString() {
    return "Id=" + id + ", fullName=" + fullName + ", address=" + address + ", avgMark=" + avgMark
            + ", math=" + math + ", literator=" + literator + ", english=" + english + ", physics=" + physics
            + ", chemistry=" + chemistry;
}
```

If we don’t write toString().
What’s happen?

Using code generator to create toString() method


a - JavaAdvance/src/basic/Student.java - Eclipse IDE
Source
Refactor
Navigate
Search
Project
Run
Window
Toggle Comment Ctrl+7
Add Block Comment Ctrl+Shift+/
Remove Block Comment Ctrl+Shift+\
Generate Element Comment Alt+Shift+J
Add Text Block Ctrl+Shift+'
Shift Right
Shift Left
Correct Indentation Ctrl+I
Format Ctrl+Shift+F
Format Element
Add Import Ctrl+Shift+M
Organize Imports Ctrl+Shift+O
Sort Members...
Clean Up...
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Generate hashCode() and equals()...
Generate toString()...
Generate Constructor using Fields...
Generate Constructors from Superclass...
Surround With Alt+Shift+Z
Externalize Strings...
Find Broken Externalized Strings



Select fields and methods to include in the toString() method:

- ☑ Fields
  - ☑ id
  - ☑ fullName
  - ☑ address
  - ☑ avgMark
  - ☑ math
  - ☑ literator
  - ☑ english
  - ☑ physics
  - ☑ chemistry

Insertion point: After 'getSchool()'

□ Generate method comments

Generated code

String format: <Default template>

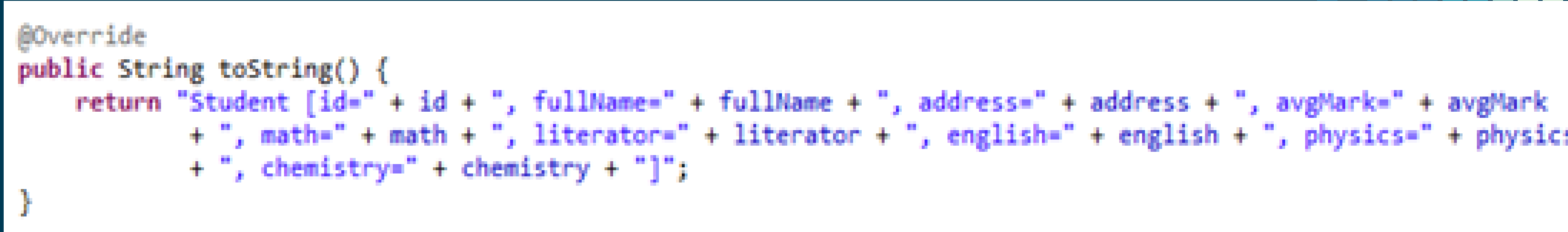
Code style: String concatenation

□ Skip null values

☑ List contents of arrays instead of using native toString()

□ Limit number of items in arrays/collections/maps to 10

9 of 9 fields and 0 of 12 methods selected.



```
@Override
public String toString() {
    return "Student [id=" + id + ", fullName=" + fullName + ", address=" + address + ", avgMark=" + avgMark
            + ", math=" + math + ", literator=" + literator + ", english=" + english + ", physics=" + physic
            + ", chemistry=" + chemistry + "]";
}
```

MAIN() TEST

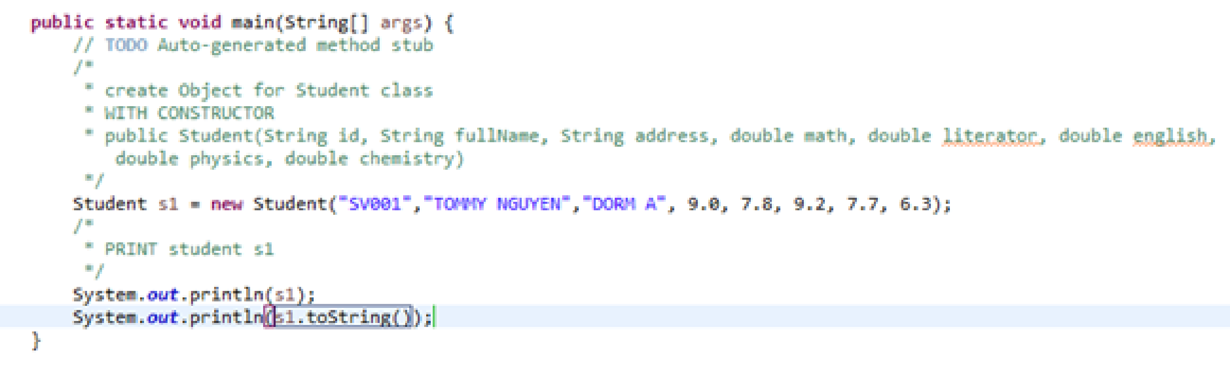
Main() Test

```
public static void main(String[] args) {
//TODO


}
```

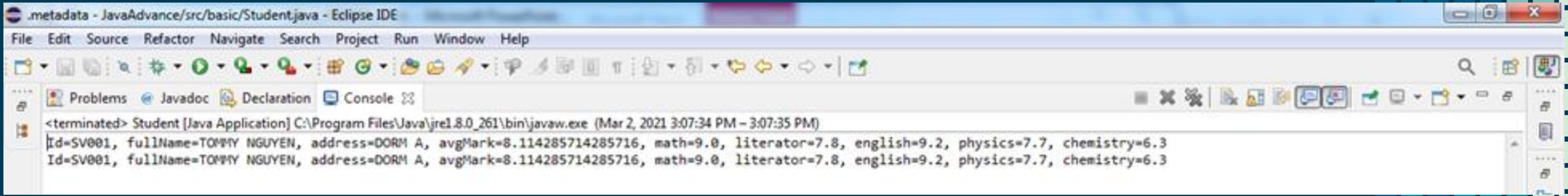
Exercise 1

Step 5: create test of Student class:

- Using difference constructors to create Student object
- *Print Student object with non-write toString, using toString() .*
- *Test some get(), set() methods*

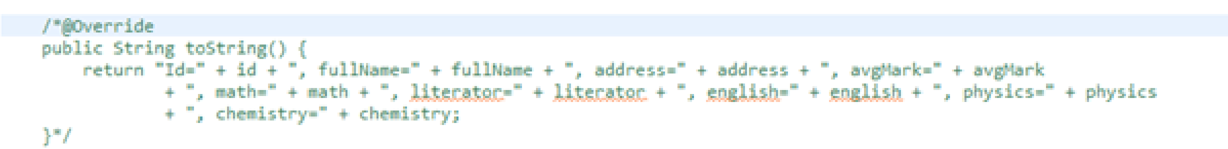


```java
public static void main(String[] args) {
    // TODO Auto-generated method stub
    /*
     * create Object for Student class
     * WITH CONSTRUCTOR
     * public Student(String id, String fullName, String address, double math, double literator, double english,
       double physics, double chemistry)
     */
    Student s1 = new Student("SV001","TOMMY NGUYEN","DORM A", 9.0, 7.8, 9.2, 7.7, 6.3);
    /*
     * PRINT student s1
     */
    System.out.println(s1);
    System.out.println(s1.toString());
}
```

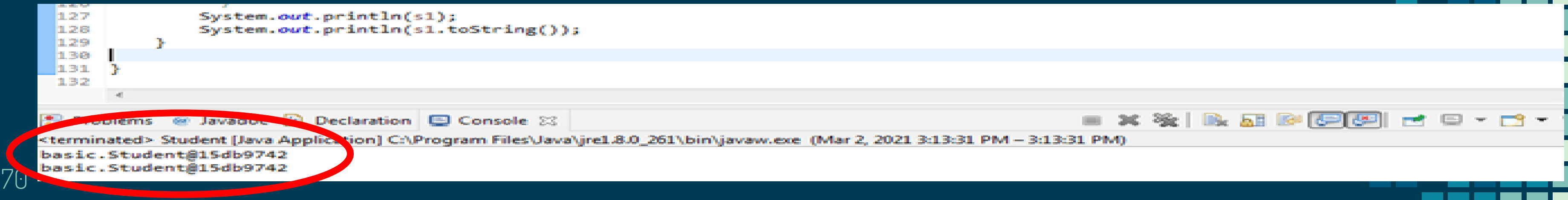


.metadata - JavaAdvance/src/basic/Student.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console

```
<terminated> Student [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (Mar 2, 2021 3:07:34 PM – 3:07:35 PM)
Id=SV001, fullName=TOMMY NGUYEN, address=DORM A, avgMark=8.114285714285716, math=9.0, literator=7.8, english=9.2, physics=7.7, chemistry=6.3
Id=SV001, fullName=TOMMY NGUYEN, address=DORM A, avgMark=8.114285714285716, math=9.0, literator=7.8, english=9.2, physics=7.7, chemistry=6.3
```

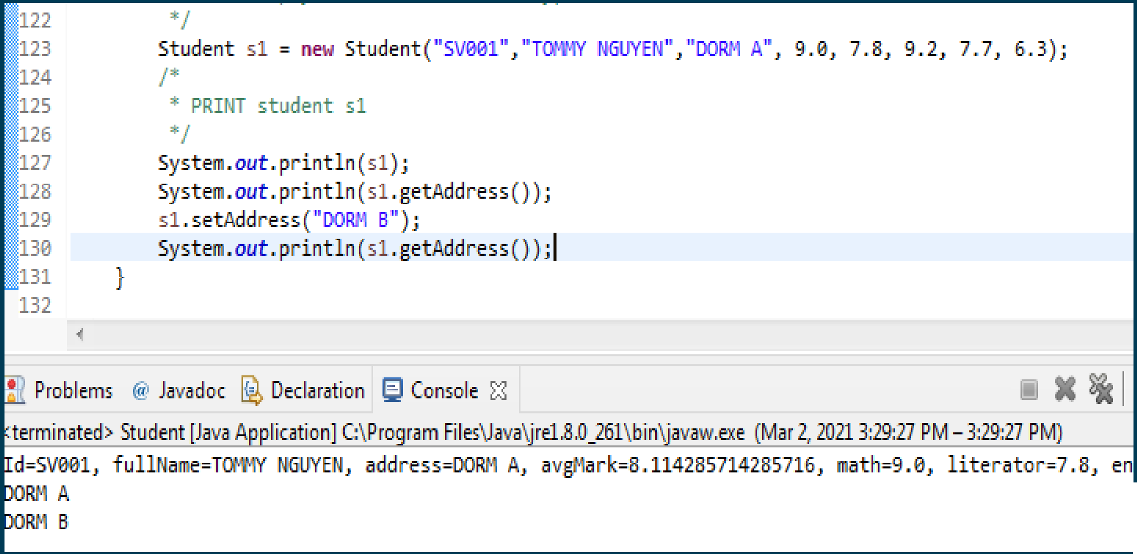


```java
/*@Override
public String toString() {
    return "Id=" + id + ", fullName=" + fullName + ", address=" + address + ", avgMark=" + avgMark
            + ", math=" + math + ", literator=" + literator + ", english=" + english + ", physics=" + physics
            + ", chemistry=" + chemistry;
}*/
```



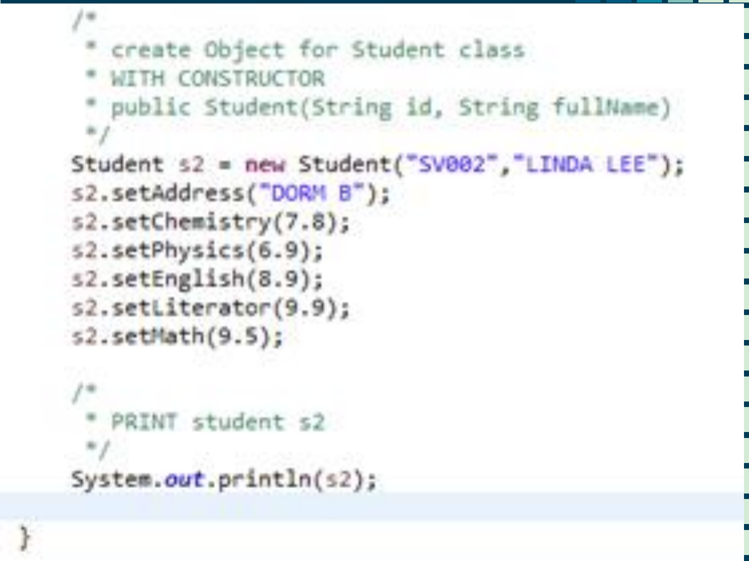
```java
            System.out.println(s1);
            System.out.println(s1.toString());
        }

}
```

```
<terminated> Student [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (Mar 2, 2021 3:13:31 PM – 3:13:31 PM)
basic.Student@15db9742
basic.Student@15db9742
```



```
         */
        Student s1 = new Student("SV001","TOMMY NGUYEN","DORM A", 9.0, 7.8, 9.2, 7.7, 6.3);
        /*
         * PRINT student s1
         */
        System.out.println(s1);
        System.out.println(s1.getAddress());
        s1.setAddress("DORM B");
        System.out.println(s1.getAddress());
    }
```

```
<terminated> Student [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (Mar 2, 2021 3:29:27 PM – 3:29:27 PM)
Id=SV001, fullName=TOMMY NGUYEN, address=DORM A, avgMark=8.114285714285716, math=9.0, literator=7.8, en
DORM A
DORM B
```



```
    /*
     * create Object for Student class
     * WITH CONSTRUCTOR
     * public Student(String id, String fullName)
     */
    Student s2 = new Student("SV002","LINDA LEE");
    s2.setAddress("DORM B");
    s2.setChemistry(7.8);
    s2.setPhysics(6.9);
    s2.setEnglish(8.9);
    s2.setLiterator(9.9);
    s2.setMath(9.5);

    /*
     * PRINT student s2
     */
    System.out.println(s2);

}
```

Exercises

Exercise 1

Question 1: create getAVG() method of Student class and test

Return: double

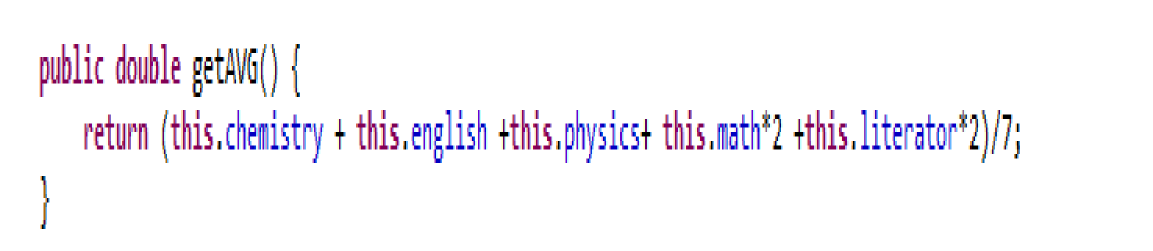
Hint:

Double mark if math and literator

How many subjects of student? ….

Sum all marks divide to quantity of subjects

*(math *2 and literator *2)*



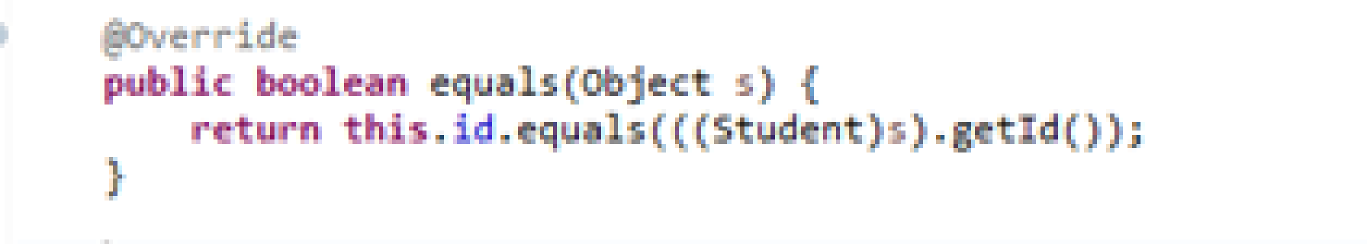
```
public double getAVG() {
    return (this.chemistry + this.english +this.physics+ this.math*2 +this.literator*2)/7;
}
```

Exercise 1

Question 2: @override

boolean equals(Object o) method of Student class and test

Return: double



```
@Override
public boolean equals(Object s) {
    return this.id.equals(((Student)s).getId());
}
```

Exercise 1

Question 3: Student class implements Comparator or Comparable to compare 2 Student object

@override compare or compareTo method

COMPARABLE

- public interface Comparable<T> {
- int compareTo(T other);
- }
- The call a.compareTo(b) must return 0 if a and b are equal, a negative integer if a comes before b in the sort order, and a positive integer if a comes after b. The exact value does not matter; only its sign (>0, 0, or < 0) matters.
- < 0 : a comes before b a <b …
- = 0 : a and b are equals a =b … hay b= a …
- > 0 : a comes after b b >a …

COMPARABLE

```
class Student implements Comparable<Student> {
            public int compareTo(Student other) {
            if (this.getAVG() > other. getAVG() )
            return 1;
            else if (this.getAVG() < other. getAVG() ) return -1;
            else
            return 0;
}
. . .
}
```

COMPARATOR

- public interface Comparator<T> {
- int compare(T a, T b);
- }
- int **compare(** Object a, Object b)
- < 0 : a comes before b -> a <b
- > 0 : a comes after b -> b > a
- = 0 : a and b are equals

COMPARATOR

```
class Student implements Comparator< Student >{
public int compare(Student a, Student b){
        if (a.getAVG() > b. getAVG() )
        return 1;
        else if (a.getAVG() < b. getAVG() ) return -1;
        else
        return 0;
}
}
```

References

- **Introduction to Java Programming** 8th , Y. Daniel Liang.
- **Data Structures and Algorithms in Java**,6th Edition, Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser

NOTES FOR NEXT WEEK

- **Review**
- **Do homework (work with computer and write down paper)**
- **Test 1 , 15 mins on class**
- String class in java collections
- Math class in java collections
- Operation (control)
- JUnit test

THANKS!