Last Lecture

Object Oriented Programming

- Classes
- Objects and Instance variables
- Constructors
- Methods

Lecture 4

Object Oriented Programming

- Class, instance and local variables
- Scope of variables

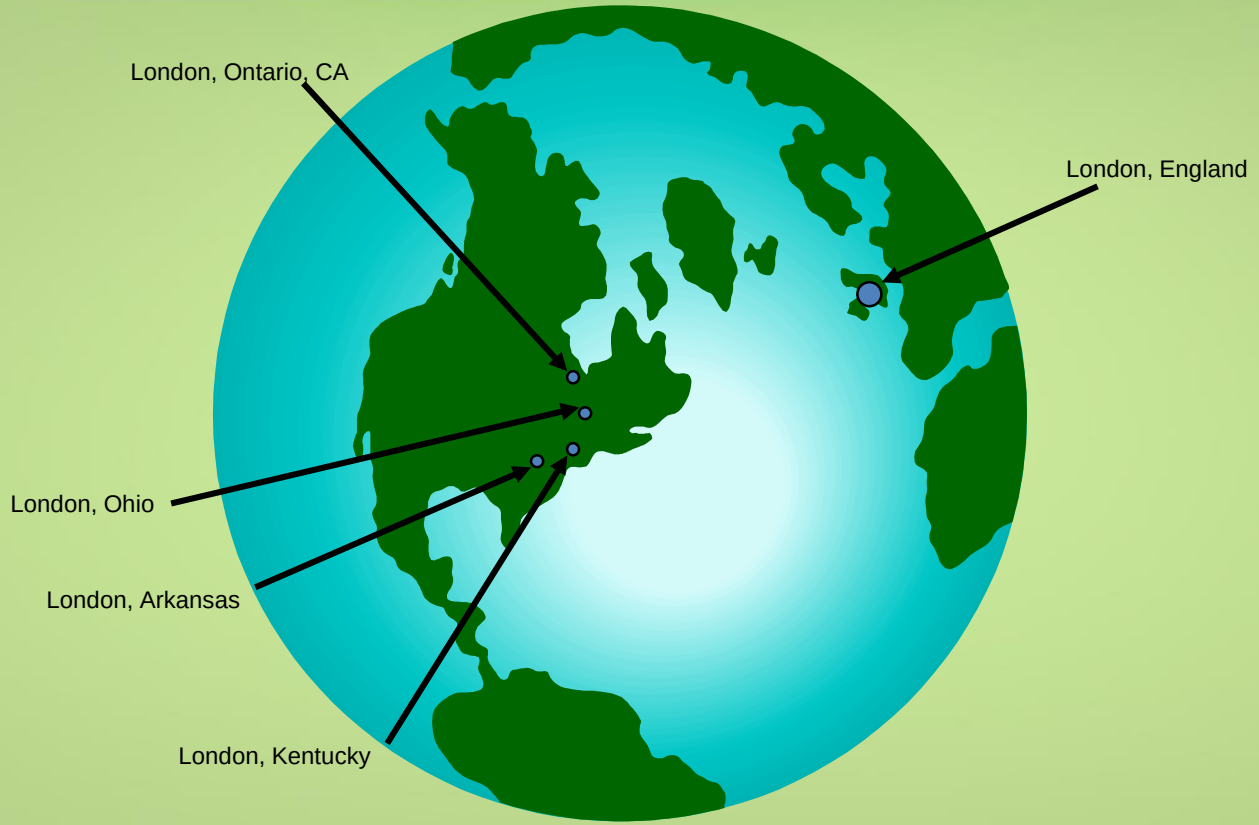
Branching statements

- if-else
- switch

What is a variable scope?

- The scope of a variable is the section of a program in which the variable can be used.
- What if we have two variables declared with the same name?

What is a variable scope?



Scope Rules for Variables

- Instance (or class) variables are accessible from anywhere within the class where they are declared.
- Local variables (including parameters) are accessible only inside the method where they are declared.
- Variables declared within a program block enclosed in a pair of curly braces { } are local to the program block.
- What if we have two variables declared with the same name?

Scope Rules for Variables

Name resolution rules for variables with the same name:

- Attempt to find a matching variable declaration in its current block
- If not found, attempt to find a matching variable declaration in its parent's block
- The process will repeat until the scope of variable is resolved
- **Compilation error** if the variable can't be resolved even in the outermost code block

An Example: Bank Account

Instance variables

```
import comp1022p.IO;
/**
 * A bank account has a balance and an owner who can make
 * deposits to and withdrawals from the account.
 */
public class BankAccount {
   private double balance = 0.0;     // Initial balance is set to zero
   private String owner = "NoName";     // Name of owner

   /**
    *   Default constructor for a bank account with zero balance
    */
   public BankAccount ( ) { }
   /**
    *   Construct a balance account with a given initial balance and owner's name
    *   @param   initialBalance     the initial balance
    *   @param   name               name of owner
    */
   public BankAccount (double initialBalance, String name)  {
     balance = initialBalance;
     owner = name;
   }
```

An Example: Bank Account

```
/**
 *   Method for depositing money to the bank account
 *   @param   dAmount     the amount to be deposited
 */
public void deposit(double dAmount)  {
    balance = balance + dAmount;
}
/**
 *   Method for withdrawing money from the bank account
 *   @param   wAmount     the amount to be withdrawn
 */
public void withdraw(double wAmount) {
    balance = balance - wAmount;
}
/**
 *   Method for getting the current balance of the bank account
 *   @return   the current balance
 */
public double getBalance() {
    return balance;
}
```

An Example: Bank Account

```
/**
 *   Method for depositing money to the bank account
 *   @param   dAmount     the amount to be deposited
 */
public void deposit(double amount)  {
    balance = balance + amount;
}
/**
 *   Method for withdrawing money from the bank account
 *   @param   wAmount     the amount to be withdrawn
 */
public void withdraw(double amount) {
    balance = balance - amount;
}
/**
 *   Method for getting the current balance of the bank account
 *   @return   the current balance
 */
public double getBalance() {
    return balance;
}
```

An Example: Bank Account

```
/**
 *   Method for depositing money to the bank account
 *   @param   dAmount     the amount to be deposited
 */
public void deposit(double amount)  {
    balance = balance + amount;
}
/**
 *   Method for withdrawing money from the bank account
 *   @param   wAmount     the amount to be withdrawn
 */
public void withdraw(double   balance  ) {
    balance = balance –  balance ;
}
/**
 *   Method for getting the current balance of the bank account
 *   @return   the current balance
 */
public double getBalance() {
    return balance;
}
```

CourseGrade example

```
/**
 * CourseGrade determines the final grade which is computed as
 * the weighted sum of the grades obtained in exam, lab and homework
 */
public class CourseGrade
{
    public static void main(String[] args)
    {
        final int examWeight = 70;    // Percentage weight given to examination
        final int labWeight = 20;     // Percentage weight given to lab work
        final int hwWeight = 10;      // Percentage weight given to homework assignment
        double examScore;          // Examination score obtained by student
        double labScore;           // Lab score obtained by student
        double hwScore;            // Homework score obtained by student
        double finalGrade;         // Final grade obtained by student
```

CourseGrade example

```
        // Ask student to input scores for exam, lab and nomework
        IO.output("Enter your exam grade: ");
        examScore = IO.inputDouble( );
        IO.output("Enter your lab grade: ");
        labScore = IO.inputDouble( );
        IO.output("Enter your homework grade: ");
        hwScore = IO.inputDouble( );

        // Computer final grade as the weighted sum of exam, lab and homework scores
        examScore = examScore * (examWeight / 100.0);
        labScore = labScore * (labWeight / 100.0);
        hwScore = hwScore * (hwWeight / 100.0);
        finalGrade = examScore + labScore + hwScore;

        // Output the final grade
        IO.outputln("Your final grade is " + finalGrade);

    }
}
```

Grade Report

Grade report for COMP1022P in Spring 2014

Name: T.C. Pong

Exam score:	100.0
Lab score:	100.0
HW score:	100.0
Final score:	100.0
Course Grade:	A

Static Variables

```
/**
 * CS101Grade captures the performance of students in a course including
 * their exam, lab and homework scores and compute the final grade as a
 * weighted sum of the exam, lab and homework scores
 */
public class CS101Grade
{
   // static and instance variables
   final int examWeight = 70;     // Percentage weight given to examination
   final int labWeight = 20;      // Percentage weight given to lab work
   final int hwWeight = 10;       // Percentage weight given to homework assignment
   double examScore;          // Examination score obtained by student
   double labScore;           // Lab score obtained by student
   double hwScore;            // Homework score obtained by student
   double finalGrade;         // Final grade obtained by student
```

Static Variables

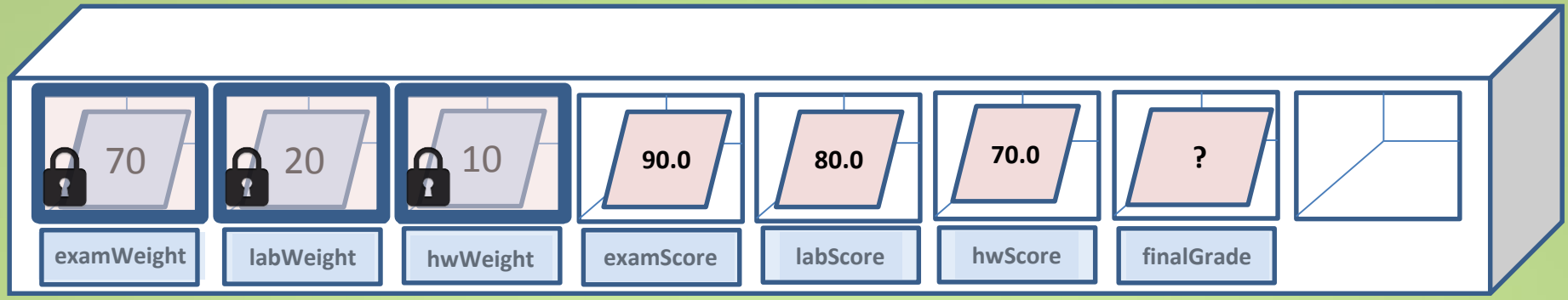
```
/**
 * CS101Grade captures the performance of students in a course including
 * their exam, lab and homework scores and compute the final grade as a
 * weighted sum of the exam, lab and homework scores
 */
public class CS101Grade
{
   // static and instance variables
   private static final int examWeight = 70; // Percentage weight given to examination
   private static final int labWeight = 20;  // Percentage weight given to lab work
   private static final int hwWeight = 10;   // Percentage weight given to homework assignment
   private double examScore;       // Examination score obtained by student
   private double labScore;        // Lab score obtained by student
   private double hwScore;         // Homework score obtained by student
   private double finalGrade;      // Final grade obtained by student
   private String studentName;     // Name of a particular student
```

```
/**
 * Constructor declaration
 */
public CS101Grade(String name) {
     studentName = name;
}

/**
 *  Method getScores obtains all scores for a student
 */
public void getScores() {
     IO.output("Enter your exam grade: ");
     examScore = IO.inputDouble( );
     IO.output("Enter your lab grade: ");
     labScore = IO.inputDouble( );
     IO.output("Enter your homework grade: ");
     hwScore = IO.inputDouble( );
}
```

```
/**
 * Compute final grade as the weighted sum of exam, lab and homework scores
 *
 * @param   examScore   Exam score of student
 * @param   labScore    Lab score of student
 * @param   hwScore     Homework score of student
 * @return  Weighted sum of examScore, labScore and hwScore in double type
 */
public double computeGrade(double examScore, double labScore, double hwScore)
{
      examScore = examScore * (examWeight / 100.0);
      labScore = labScore * (labWeight / 100.0);
      hwScore = hwScore * (hwWeight / 100.0);

      return examScore + labScore + hwScore;
}
```

```
    /**
     * Set the finalGrade by calling computeGrade
     */
     public void setFinalGrade(){
         finalGrade = computeGrade(examScore, labScore, hwScore);
     }

     /**
      * Output the final results
      */
     public void outputResult(){
        IO.outputln("For " + studentName + ": examScore = " + examScore +
                    " labScore = " + labScore + " hwScore = " + hwScore +
                    " finalGrade = " + finalGrade);
     }
}
```

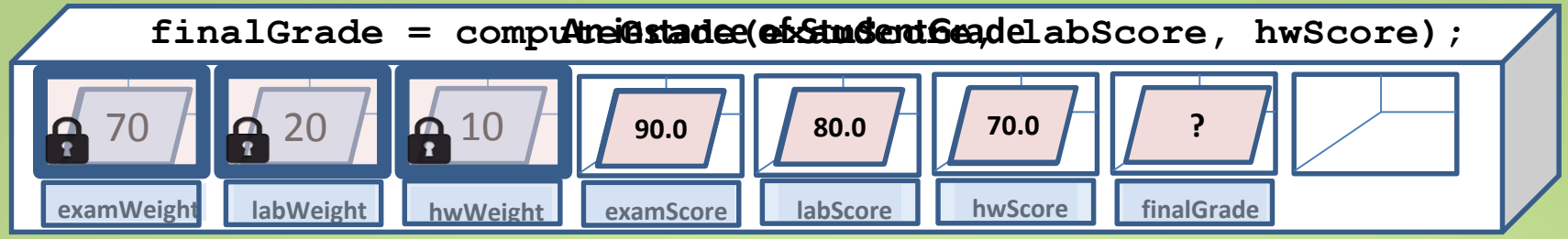
Memory allocation for variables



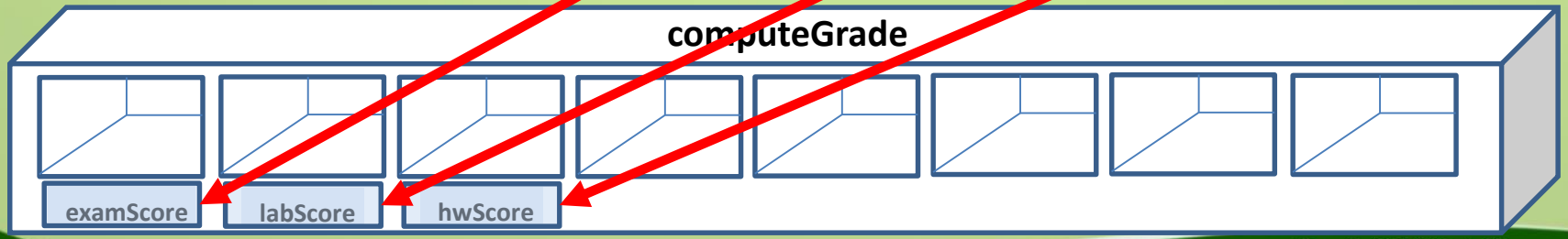

```
final int examWeight = 70;
final int labWeight  = 20;
final int hwWeight  = 10;
```

```
double examScore;
double labScore;
double hwScore;
double finalGrade;
```

Method: computeGrade

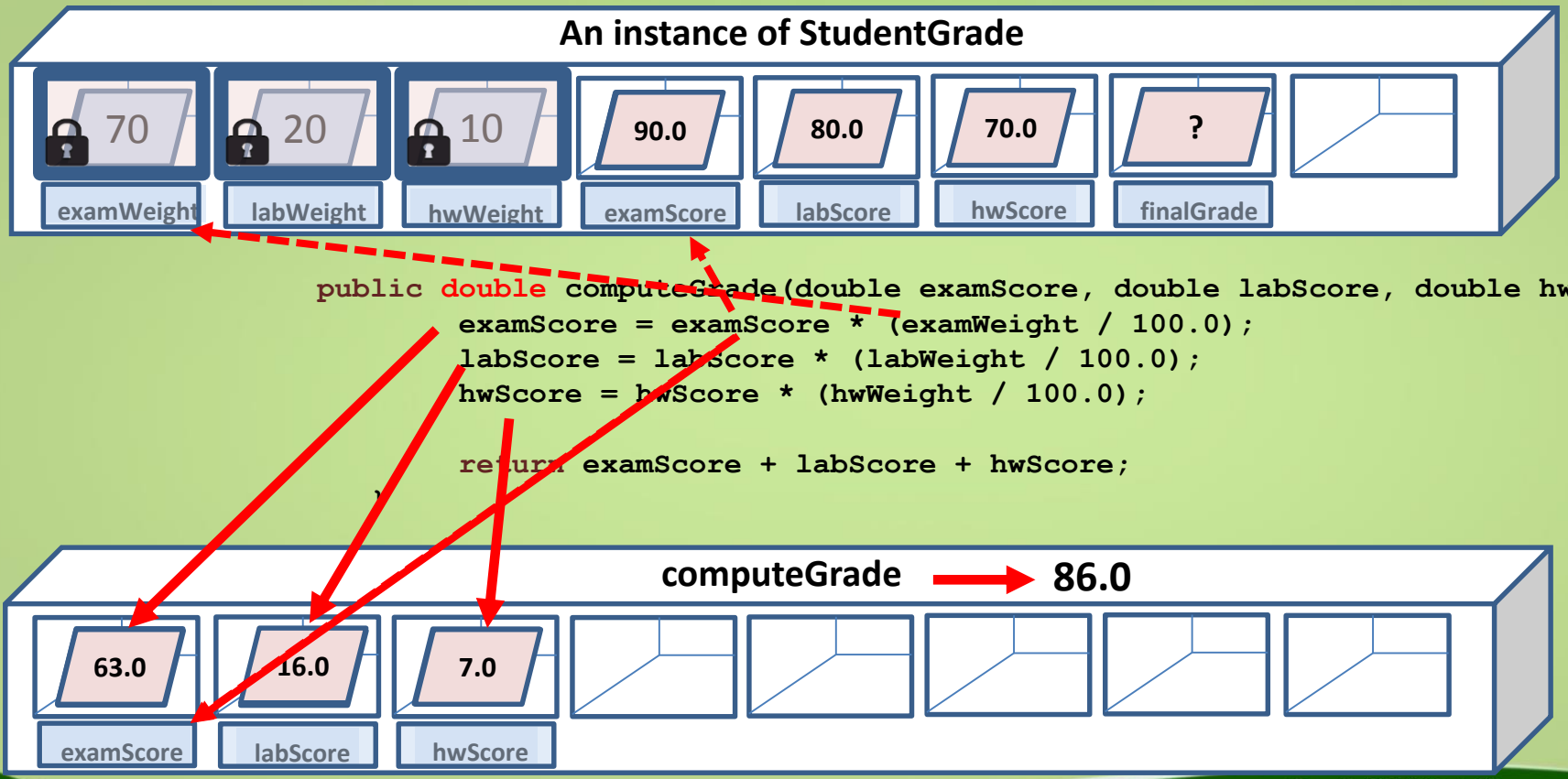


```
public double computeGrade(double examScore, double labScore, double hwScore) {
        examScore = examScore * (examWeight / 100.0);
        labScore = labScore * (labWeight / 100.0);
        hwScore = hwScore * (hwWeight / 100.0);

        return examScore + labScore + hwScore;
    }
```



香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Method: computeGrade

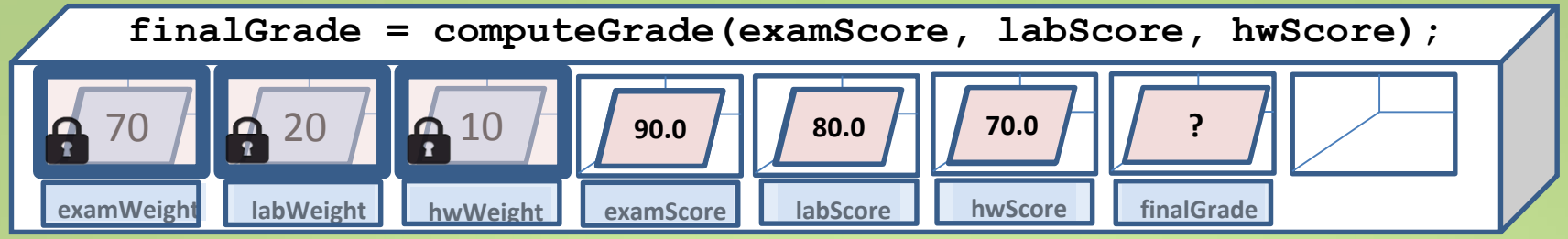


```
/**
 * Compute final grade as the weighted sum of exam, lab and homework scores
 *
 * @param   examScore   Exam score of student
 * @param   labScore    Lab score of student
 * @param   hwScore     Homework score of student
 * @return  Weighted sum of examScore, labScore and hwScore in double type
 */
public double computeGrade(double examScore, double labScore, double hwScore)
{
      examScore = examScore * (examWeight / 100.0);
      labScore = labScore * (labWeight / 100.0);
      hwScore = hwScore * (hwWeight / 100.0);

      return examScore + labScore + hwScore;
}
```

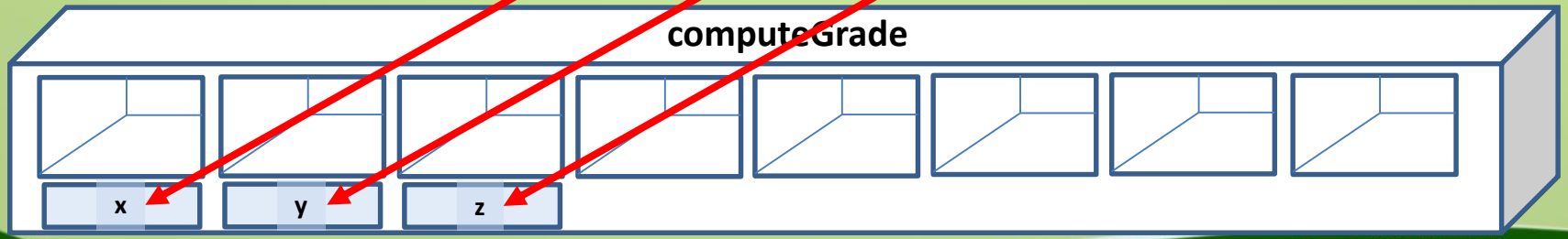
```
/**
 * Compute final grade as the weighted sum of exam, lab and homework scores
 *
 * @param   examScore   Exam score of student
 * @param   labScore    Lab score of student
 * @param   hwScore     Homework score of student
 * @return  Weighted sum of examScore, labScore and hwScore in double type
 */
public double computeGrade(double x, double y, double z)
{
      x = x * (examWeight / 100.0);
      y = y * (labWeight / 100.0);
      z = z * (hwWeight / 100.0);

      return x + y + z;
}
```

```
/**
 * Compute final grade as the weighted sum of exam, lab and homework scores
 *
 * @param   examScore   Exam score of student
 * @param   labScore    Lab score of student
 * @param   hwScore     Homework score of student
 * @return  Weighted sum of examScore, labScore and hwScore in double type
 */
public double computeGrade(double x, double y, double z)
{
      examScore = x * (examWeight / 100.0);
      labScore = y * (labWeight / 100.0);
      hwScore = z * (hwWeight / 100.0);

      return examScore + labScore + hwScore;
}
```

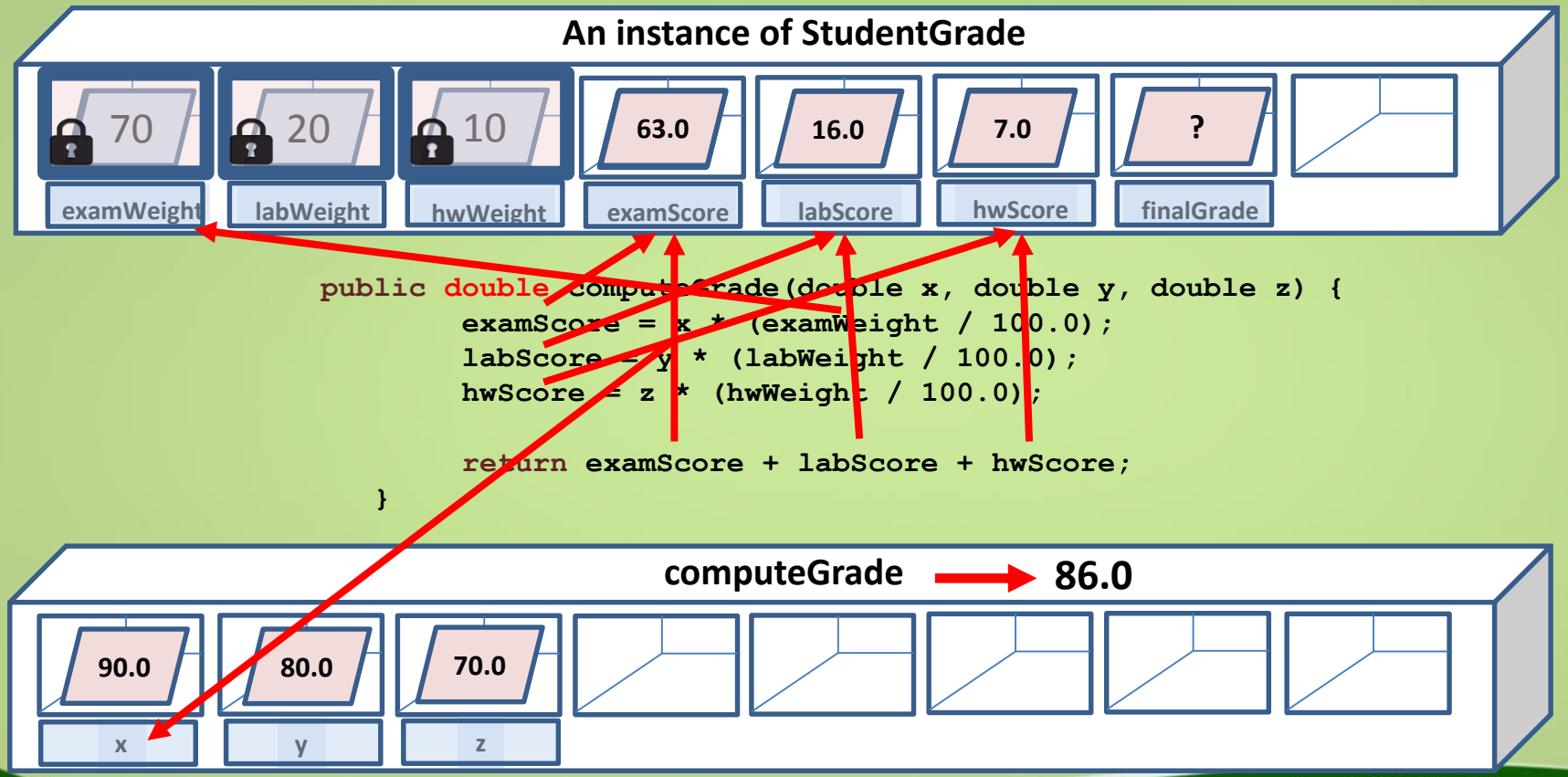
Method: computeGrade




```
public double computeGrade(double x, double y, double z) {
        examScore = x * (examWeight / 100.0);
        labScore = y * (labWeight / 100.0);
        hwScore = z * (hwWeight / 100.0);

        return examScore + labScore + hwScore;
    }
```

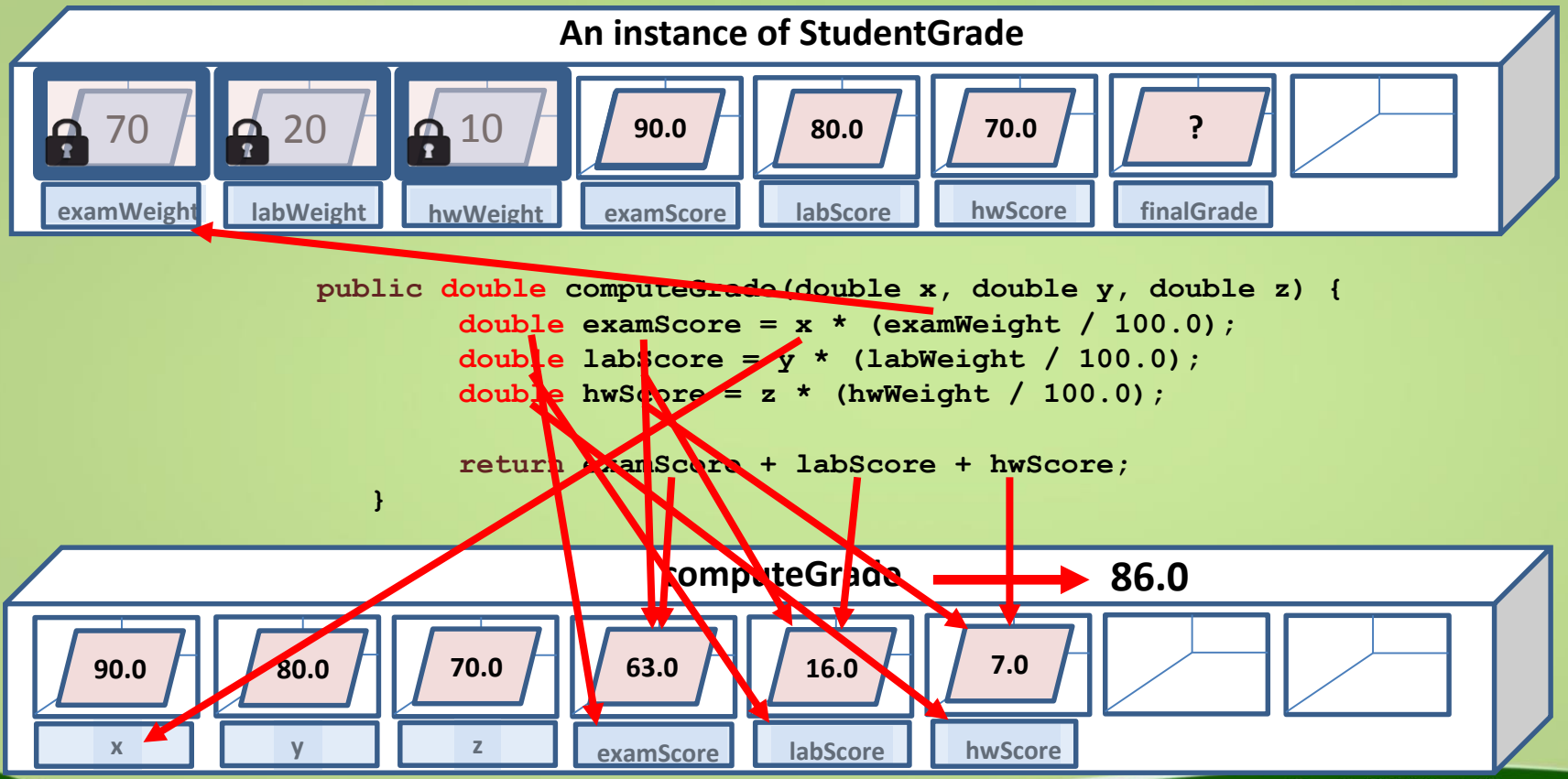


Method: computeGrade



```
/**
 * Compute final grade as the weighted sum of exam, lab and homework scores
 *
 * @param   examScore   Exam score of student
 * @param   labScore    Lab score of student
 * @param   hwScore     Homework score of student
 * @return  Weighted sum of examScore, labScore and hwScore in double type
 */
public double computeGrade(double x, double y, double z)
{
      double examScore = x * (examWeight / 100.0);
      double labScore = y * (labWeight / 100.0);
      double hwScore = z * (hwWeight / 100.0);

      return examScore + labScore + hwScore;
}
```

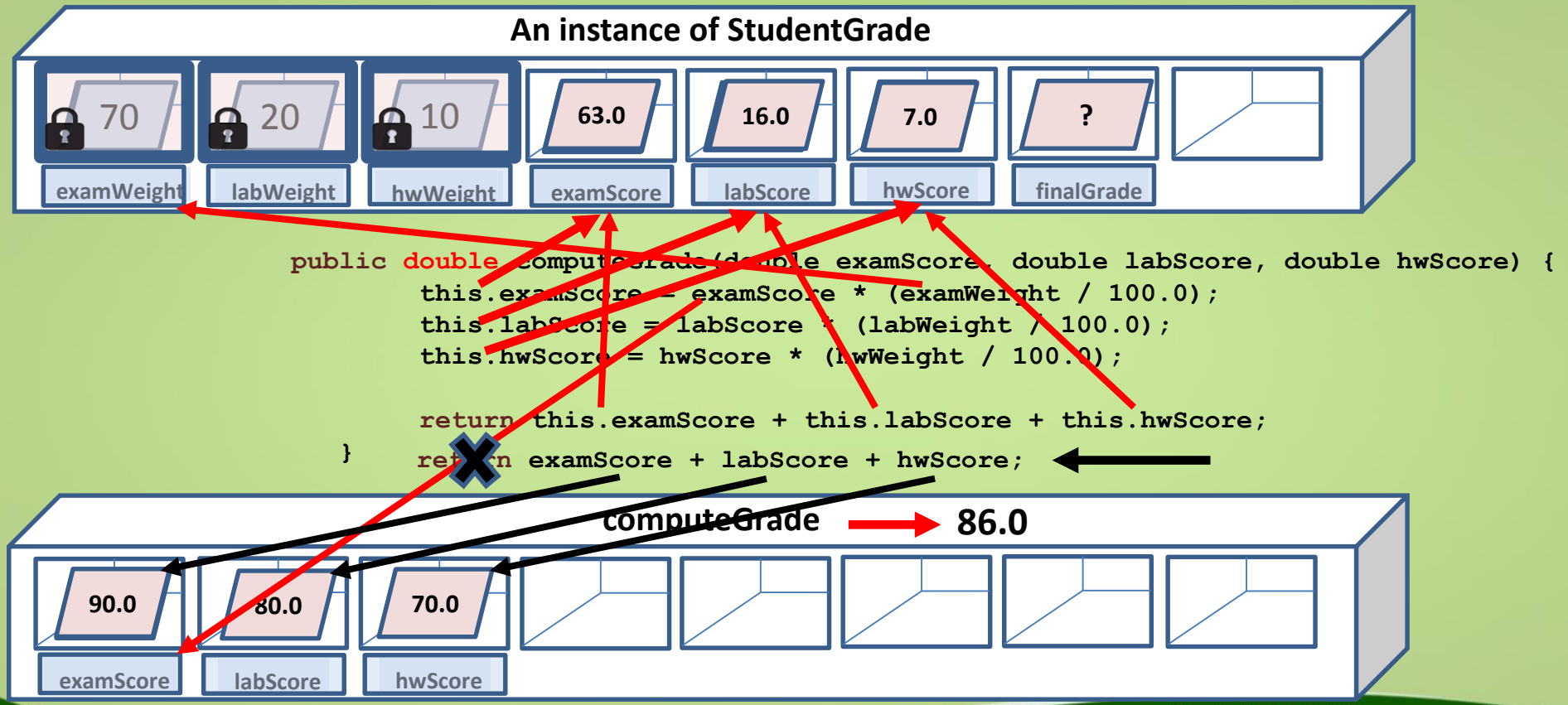
Method: computeGrade



The “this” keyword

```
/**
 * Compute final grade as the weighted sum of exam, lab and homework scores
 *
 * To illustrate the use of the keyward “this” which can be used to make
 * reference to the current object, that is, the object whose method or
 * constructor is being called.
 *
 */
public double computeGrade(double examScore, double labScore, double hwScore)
{
      this.examScore = examScore * (examWeight / 100.0);
      this.labScore = labScore * (labWeight / 100.0);
      this.hwScore = hwScore * (hwWeight / 100.0);

      return this.examScore + this.labScore + this.hwScore;
}
```

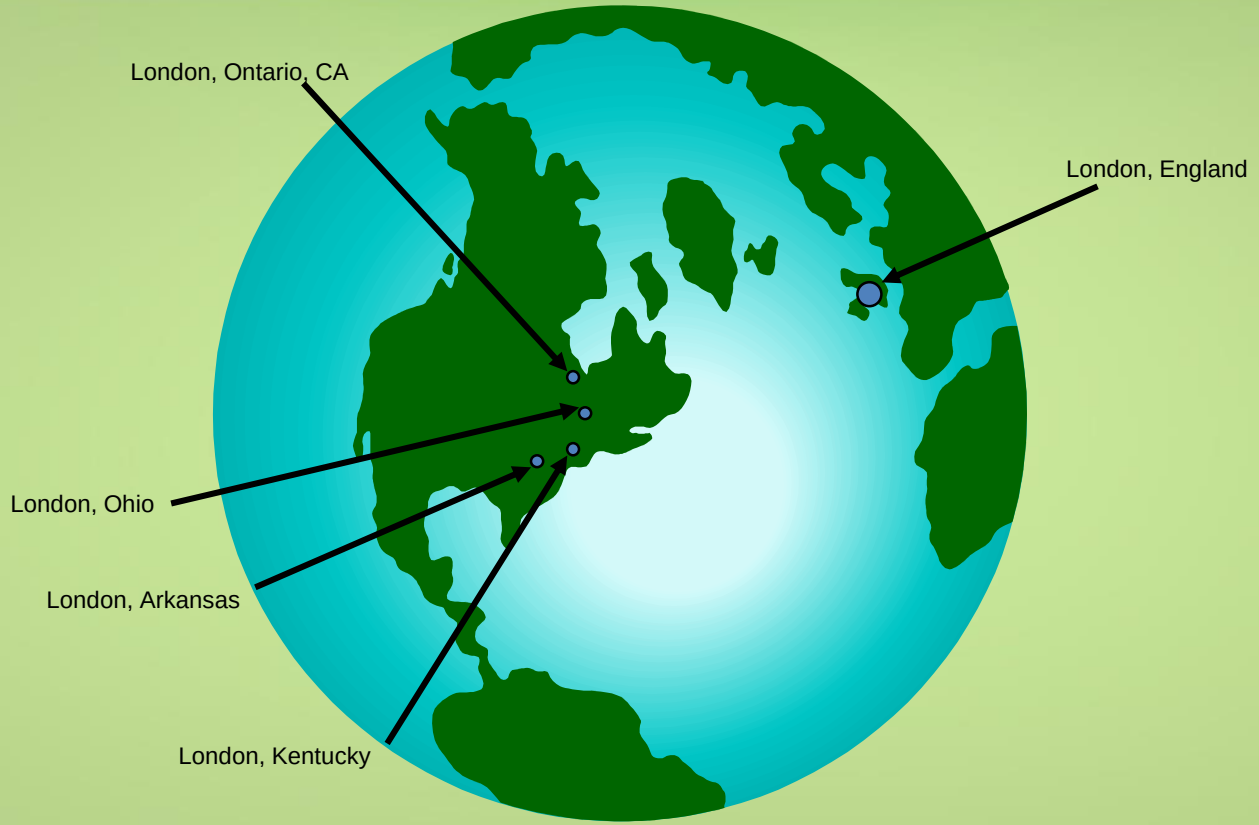
The “this” keyword



What is a variable scope?

- Scope of a variable is the region by which the variable can be used
- A variable's scope is enclosed by the **closest** pair of curly braces { } that enclose the variable declaration
- What if we have two variables declared with the same name?

What is a variable scope?



What is a variable scope?

- Scope of a variable is the region by which the variable can be used
- A variable's scope is enclosed by the **closest** pair of curly braces { } that enclose the variable declaration
- What if we have two variables declared with the same name?
  - Attempt to find a matching variable declaration in its current block
  - If not found, attempt to find a matching variable declaration in its parent's block
  - The process will repeat until the scope of variable is resolved
  - Compilation error occurs if the variable can't be resolved even in the outermost code block

The Car Example

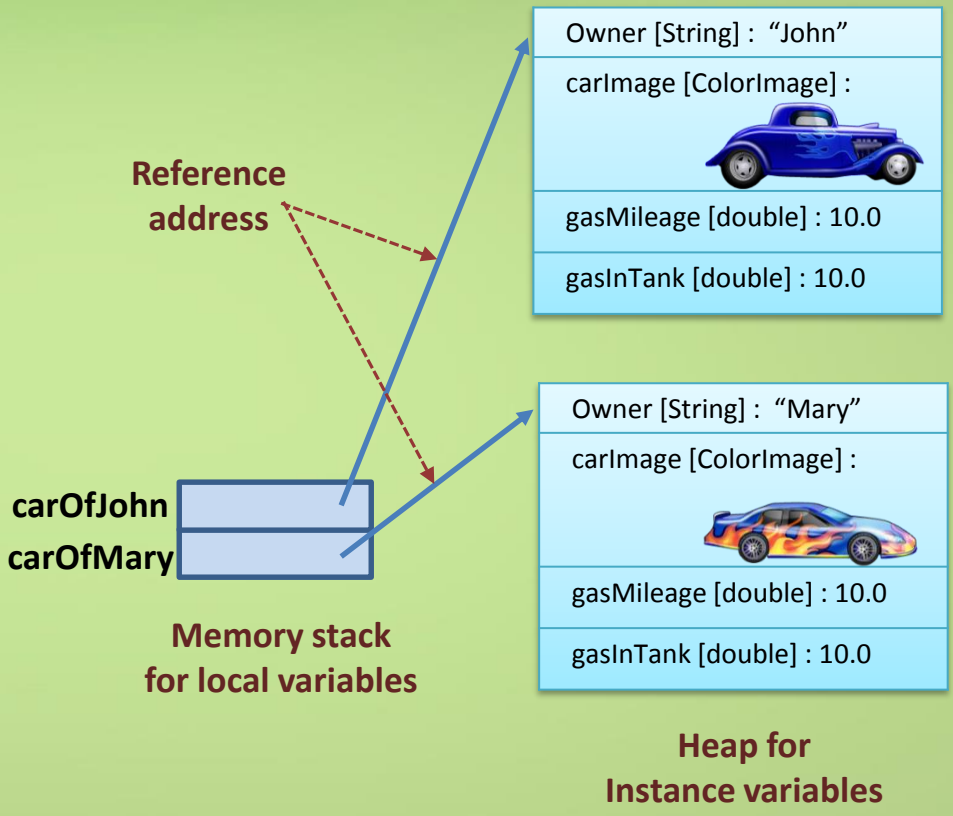
```
//   A class of Car objects that can move forward, backward and turn
public class Car2
{

    String owner = "NoName";
    ColorImage carImage = new ColorImage("Car1.png");
    double gasMileage = 10.0;  // Liters of gas used for every 100km
    double gasInTank = 10.0;
```

Instance variables

Class Car2
Owner [String] : "NoName"
carImage [ColorImage] : 
gasMileage [double] : 10.0
gasInTank [double] : 10.0

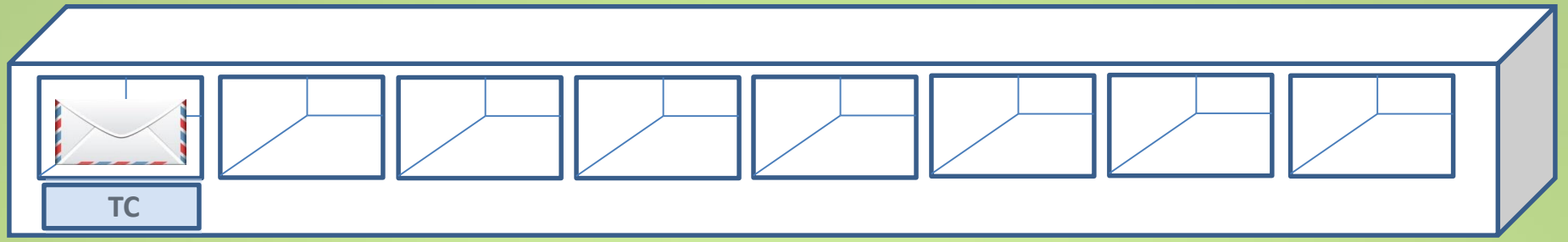
The Car Example

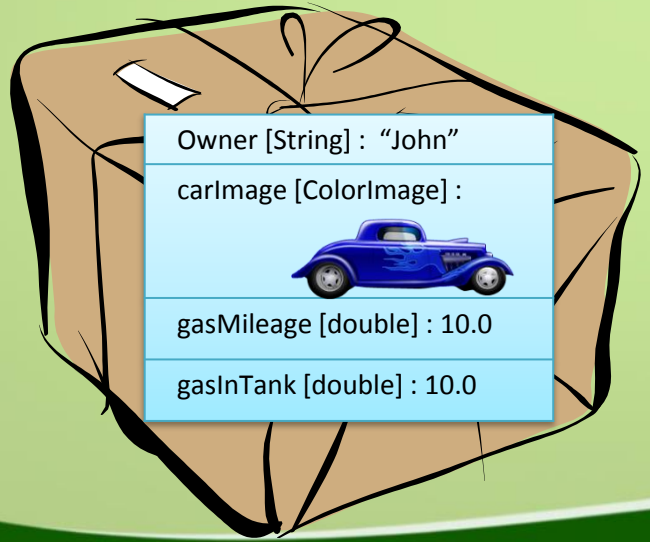
```
// Constructor for Car2
public Car2(String nameOfOwner)
  {
     owner = nameOfOwner;
     carImage = new ColorImage();
  }

public static void main(String[ ] args) {
     Car2 carOfJohn = new Car2("John");
     Car2 carOfMary = new Car2("Mary");
```

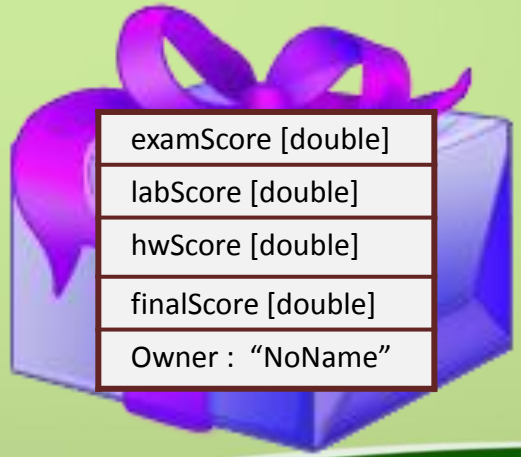


Primitive types and Reference types

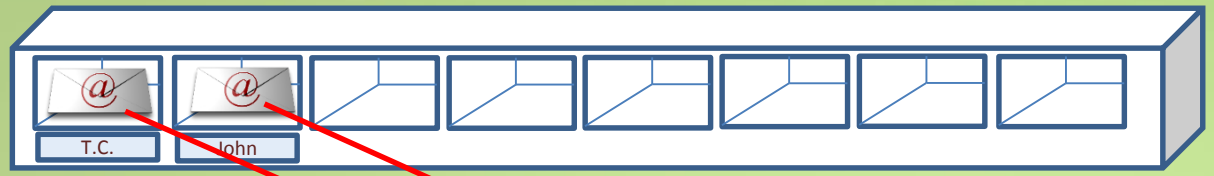
For primitive types, mailbox was used as an analogy for memory space allocated.





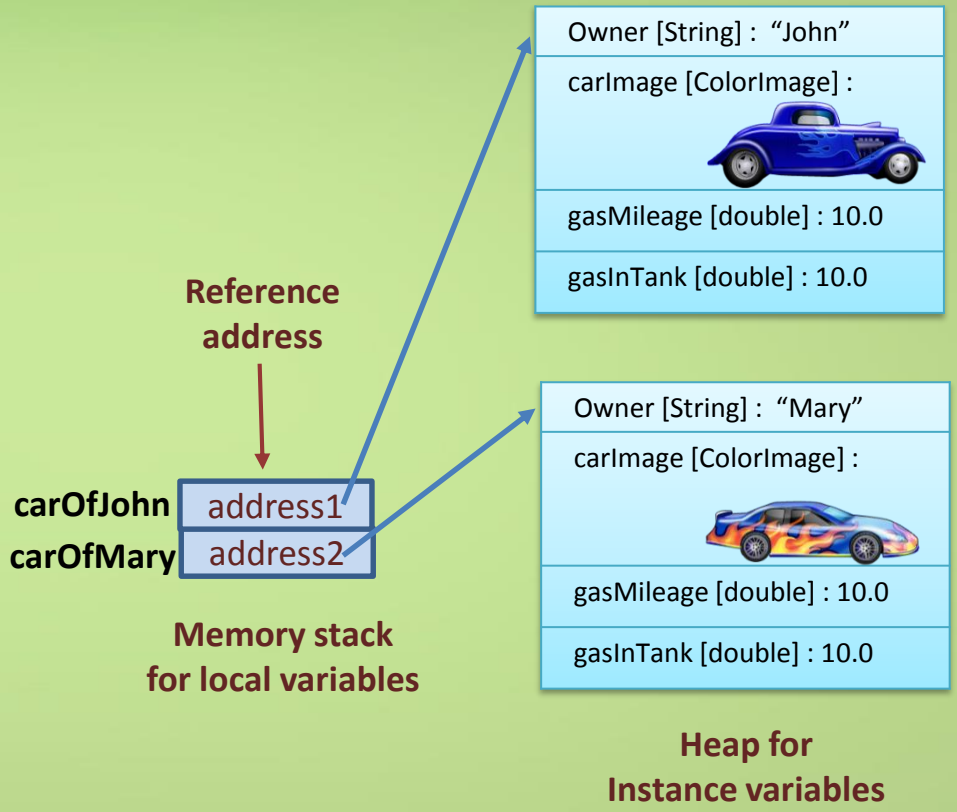


Reference types



The Car Example

```
// Constructor for Car2
public Car2(String nameOfOwner)
  {
     owner = nameOfOwner;
     carImage = new ColorImage();
  }

public static void main(String[ ] args) {
     Car2 carOfJohn = new Car2("John");
     Car2 carOfMary = new Car2("Mary");
```



Class, Instance and Local Variables

	Local variable	Instance variable	Class / static variable
Declaration	Inside a method/constructor or as formal parameters	Inside a class, but outside methods/constructors	Inside a class with 'static', but outside methods/constructors
Usage and Lifetime	Used by methods to hold intermediate results. Created when the method is entered and destroyed on exit.	A separate copy is created for each object using the keyword new and destroyed when no more reference is made by any object.	One single copy of variable shared by all objects is created when the program starts and destroyed when the program ends.

Example: Class of Car2	Public void moveForward(int dist) {
	….
	double gasUsed =
	Math.abs(dist) / 100.0 * gasMileage;
		String owner = "NoName";
		ColorImage carImage =
		new ColorImage("Car1.png");
		double gasMileage = 10.0;
			static final int NUMOFWHEEL = 4;

Initialization	**Must** be initialized before use.	Initialized to default values. E.g. 0 for numbers and null for objects.	Initialized to default values. E.g. 0 for numbers and null for objects.
Scope and Visibility	Visible only in the method/constructor or block in which they are declared. Access modifiers cannot be used.	Can be seen by all methods /constructors in the class. Visibility to other classes depends on access modifiers. They are often declared a private to protect them from being accidentally changed.	Same as instance variables but often declared as constants.

Boolean Expressions

Boolean expressions are similar to arithmetic expressions except:

- Boolean type is declared using the keyword boolean
- Boolean variables have only two possible values: true or false
- Arithmetic operators are now replaced by relational operators and conditional operators
- Examples:
  - `gasInTank == 0` is true if the value in **gasInTank** is equal to 0, and is false otherwise
  - `examScore <= 100.0` is true if the value in **examScore** is less than 100.0, and is false otherwise
  - (**examScore** >= 0.0) && (**examScore** <= 100.0) returns true if **examScore** is within the range from 0.0 to 100.0 and false otherwise
  - Mathematical expression 0 < examScore < 100 not a legal Java expression.

Relational operators

Operator	Name	Example
<	Less than	2 < 3
<=	Less than or equal to	2 <= 3
>	Greater than	2 > 3
>=	Greater than or equal to	2 >= 3
==	Equal to (Note that, you must have two consecutive '=')	2 == 3
!=	Not equal to	2 != 3

Conditional operators

- They are used to connect multiple boolean expressions

Operator	Name	Example
!	Logical NOT	! (2 == 3)
&&	Logical AND	4 < 5 && 2 < 3
||	Logical OR	4 < 5 || 2 < 3

Truth Table for Logical NOT

- Truth table
  - A mathematical table used in logic to compute the values of logical expressions
- Truth table for Logical NOT
  - For example: if **p** represents a false boolean expression, **!p** represents a true boolean expression

p	**!p**
false	true
true	false

Truth Table for Logical AND

- If both **P** and **Q** are true (i.e. both represent true boolean expressions), **P && Q** is true
- Otherwise, **P && Q** is false

P	**Q**	**P && Q**
false	false	false
false	true	false
true	false	false
true	true	true

- E.g. (**examScore** >= 0.0) && (**examScore** <= 100.0)

Truth Table for Logical OR

- If both **P** and **Q** are false (i.e. both represent false boolean expressions), **P || Q** is false
- Otherwise, **P || Q** is true

P	**Q**	**P || Q**
false	false	false
false	true	true
true	false	true
true	true	true

- E.g. (COMP1022PGrade == 'A' || COMP1021Grade == 'A') are good programmers.

More Operator Precedence

- Precedence of operators (from highest to lowest)
  - Logical not `!`
  - Multiplicative operators `* / %`
  - Additive operators `+ -`
  - Relational ordering `< <= >= >`
  - Relational equality `== !=`
  - Logical and `&&`
  - Logical or `||`
  - Assignment `=`

Branching Statements

- Motivation
  - A computer program should be smart enough to make decisions and behave differently depending on the situation
  - Example
    - Taking a lift/elevator in a building
    - Taking alternative action when you car is running out of gas

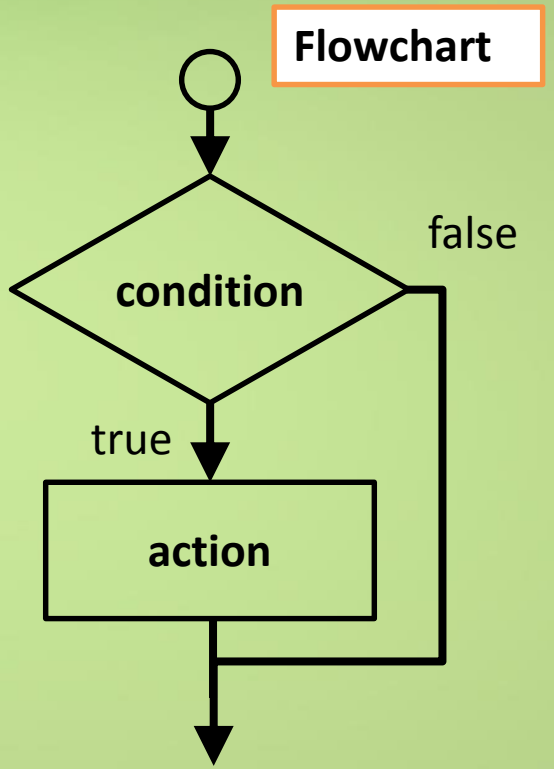
Types of branching statements

- There are three different types of branching statements
  - **if** statements
  - **if-else** statements
  - **switch** statements
- General format :
  - **Condition part** checks if the condition (usually a boolean expression) is satisfied
  - **Action part** performs the action if the condition is satisfied

if statements

```
if ( boolean-expression) {
   // if block
   statement(s);
}
```

Note: the curly brackets { } can be omitted if there is only one statement



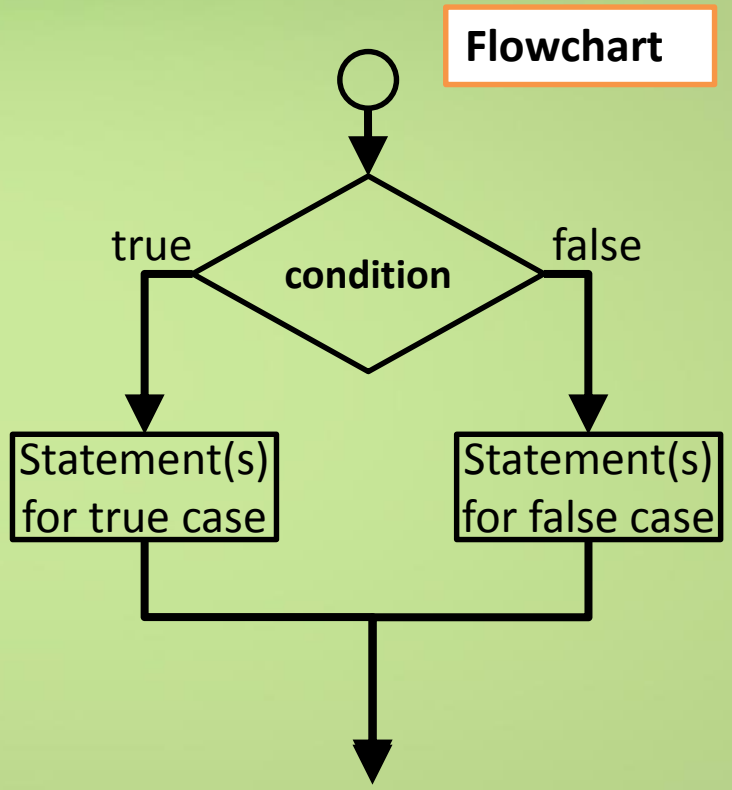
Example : Absolute value

```
/**
 * The method absolute takes an integer
 * parameter & returns its absolute value
 *
 * @param value  argument whose absolute value is
 *                to be determined
 * @return       absolute value of the argument
 */
public int absolute (int value)
{
    if(value < 0)
       value = -value;

    return value;
}
```

if-else statements

```
if ( boolean-expression ) {
  //if-block
  statement(s)-for-true-case;
}
else {
  //else-block
  statements(s)-for-false-case;
}
```

Note: the curly brackets { } can be omitted if there is only one statement in either block



Example : Find the Larger One

```
/**
 * The method larger returns the larger of two
 * integer parameters
 *
 * @param value1  first argument for comparison
 * @param value2  second argument for comparison
 * @return        larger of the two arguments
 */
public int larger (int value1, int value2)
{
    if(value1 > value2)
       return value1;
    else
       return value2;
}
```

Example : moveForward

```
public void moveForward(int dist)
{
    /*
     * Need to check if there is enough gas to travel the given
     * distance dist
     */

    // move the car by a distance dist in the orientation of the car
    double radian = Math.toRadians(carImage.getRotation());
    double distX = dist * Math.cos(radian);
    double distY = dist * Math.sin(radian);
    carImage.setX(carImage.getX() + (int)distX);
    carImage.setY(carImage.getY() + (int)distY);

    double gasUsed = Math.abs(dist) * gasMileage / 100.0;
    gasInTank = gasInTank - gasUsed;
    IO.outputln("Gas used:" + gasUsed + ",gas remained:" + gasInTank);
}
```

Example : moveForward

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)

  dist = maxDist;
```

Example : moveForward

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)

  if (dist < 0) dist = - maxDist;

  else dist = maxDist;
```

Example : moveForward

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)

   if (dist < 0) dist = - maxDist;

   else dist = maxDist;
```

Example : moveForward

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)

  if (dist < 0) dist = - maxDist;

else dist = maxDist;
```

Example : moveForward

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)
{
  if (dist < 0) dist = - maxDist;
  else dist = maxDist;

  IO.outputln("Not enough gas to complete the trip!");
}
```

Example : moveForward

```
public void moveForward(int dist)
{
    int maxDist = (int) (gasInTank / gasMileage * 100.0);
    if (Math.abs(dist) > maxDist)
    {
      if (dist < 0) dist = - maxDist;
      else dist = maxDist;
      IO.outputln("Not enough gas to complete the trip!");
    }

    // move the car by a distance dist in the orientation of the car
    double radian = Math.toRadians(carImage.getRotation());
    double distX = dist * Math.cos(radian);
    double distY = dist * Math.sin(radian);
    carImage.setX(carImage.getX() + (int)distX);
    carImage.setY(carImage.getY() + (int)distY);

    double gasUsed = Math.abs(dist) * gasMileage / 100.0;
    gasInTank = gasInTank - gasUsed;
    IO.outputln("Gas used:" + gasUsed + ",gas remained:" + gasInTank);
}
```

Dangling-else problem

```
int a=10, b=5, c=10;
if(a>b)
   if(b>c)
         a = 20;
else
   a = 30;
```

?

```
int a=10, b=5, c=10;
if(a>b)
   if(b>c)
         a = 20;
else
         a = 30;
```

?

Question: The only difference between the two code blocks is the indentation on the else-clause.

- Which if-statement does the else-clause belong to?
- What would be the value of **a**?

Dangling-else problem

```
int a=10, b=5, c=10;
if(a>b)
if(b>c)
   a = 20;
else
   a = 30;
```

the same as

```
int a=10, b=5, c=10;
if(a>b) {
   if(b>c)
        a = 20;
   else
        a = 30;
}
```

- Indentation takes no effect.
- The else-clause is paired with the closest possible if-clause.
- Use braces to make your intention clear!
- a = 30 is the correct result.

if-else-if Statement

- Example:

```
if(score >= 90)
     Grade = 'A';
else if(score >= 80)
     Grade = 'B';
else if(score >= 70)
     Grade = 'C';
else if(score >= 50)
     Grade = 'D';
else
     Grade = 'F';
```

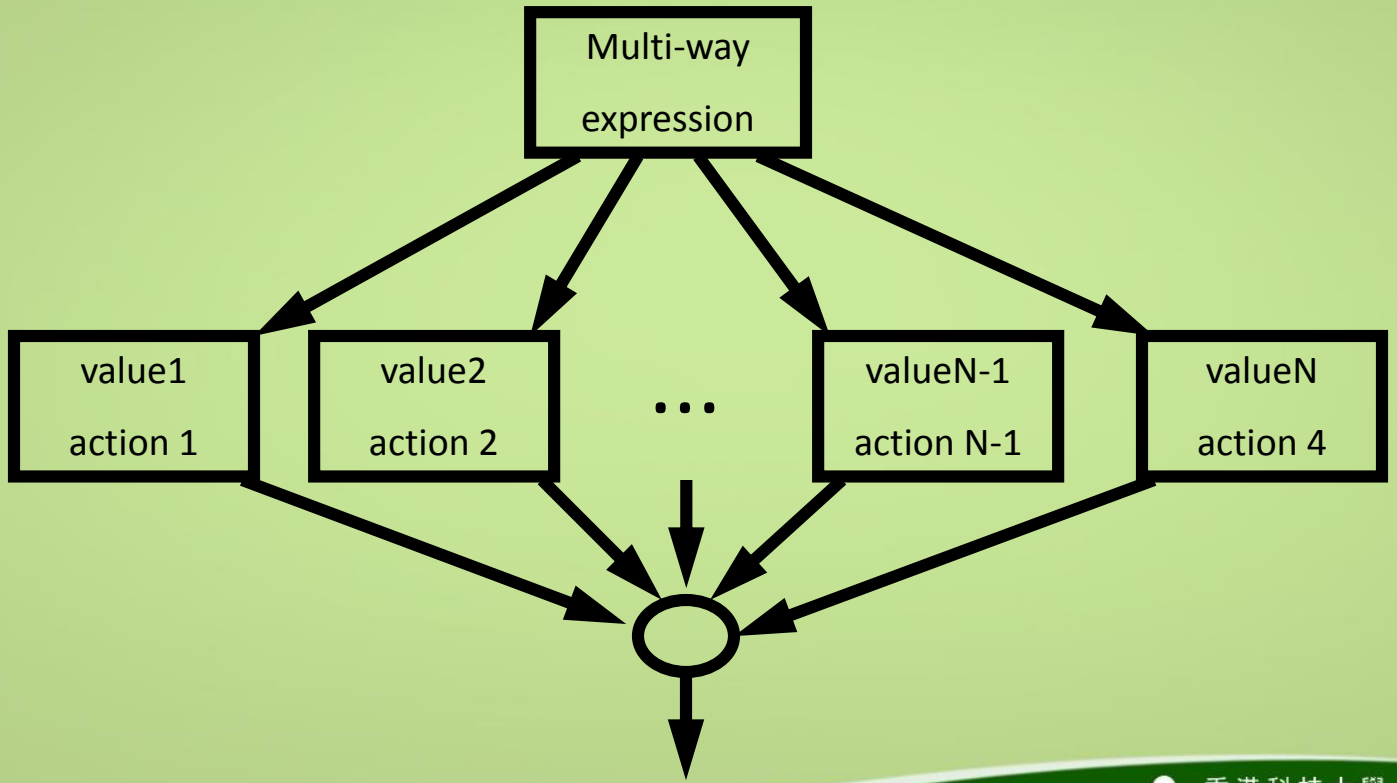
if-else-if Statement

- Example:

```
if(score >= 90)
     Grade = 'A';
if(score >= 80)
     Grade = 'B';
if(score >= 70)
     Grade = 'C';
if(score >= 50)
     Grade = 'D';
else
     Grade = 'F';
```

Switch statement

The switch statement allows more than two execution paths.

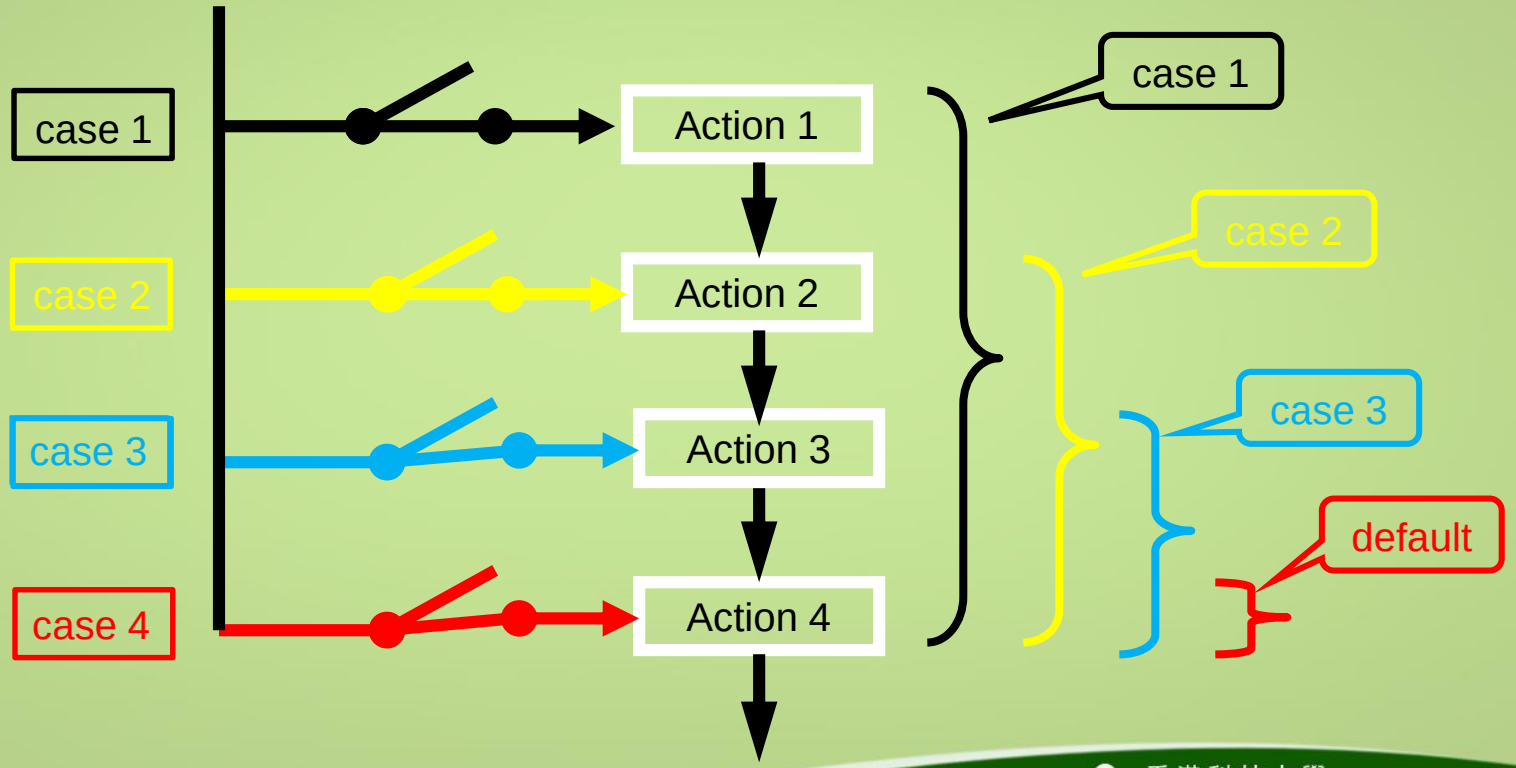


Switch statement

- Switch expression
  - Must be a value of **char**, **byte**, **short**, **int** or **String** type
- The value1, value2...valueN
  - must be of the **same type** as the switch expression
- The keyword **break** is used to exit the switch statement
  - Without the keyword break, the flow moves to the next case until a break is met
- Default (optional)
  - Only be executed when no other case is matched

```
switch (switch-expression) {
  case value1: statement(s)1;
 /* If you forget the "break;" here, the flow moves to
     the next case until a break is met */
       break;
   case value2: statement(s)2;
       break;
  ...
  case valueN: statement(s)N;
       break;
  /* default case is optional */
  default: statement(s)-for-default;
}
```

Switch statement



Switch Statement

- Example:

```
switch(score/10) {
    case 10: //next action
    case 9: grade = 'A';
            break;
    case 8: grade = 'B';
            break;
    case 7: grade = 'C';
            break;
    case 6: //next action
    case 5: grade = 'D';
            break;
    default:grade = 'F';
}
```

- Example:

```
if(score >= 90)
        Grade = 'A';
else if(score >= 80)
        Grade = 'B';
else if(score >= 70)
        Grade = 'C';
else if(score >= 50)
        Grade = 'D';
else
        Grade = 'F';
```