

MODULE 3, 4

Module 3: Decision-Making and Iterations

Module 4: Introducing Classes

Module 3: Decision-Making and Iterations

- Decision – Making Statements
- Introduction to Loops
- Jump Statements



Module 3: Decision – Making Statements

- Enable us to change the flow of the program based on the evaluation of the conditions.
- They are :
 - if
 - if – else - if
 - switch case

if Statement

- The `if-else` statement tests the result of a condition, and performs appropriate actions based on the result.

```
if (condition)
{
    //one or more statements;
}
```

if else Statement

- It can be used to route program execution through two different paths.

```
if (condition)
{
    //one or more statements;
}
else
{
    //one or more statements;
}
```

Multiple if Statement

- Known as if else if ladder.
- The condition is evaluated sequentially starting from the top of the ladder and moving downwards.

```
if (condition){  
    //one or more statements;  
}  
else if {  
    //one or more statements;  
}  
else {  
    //one or more statements;  
}
```

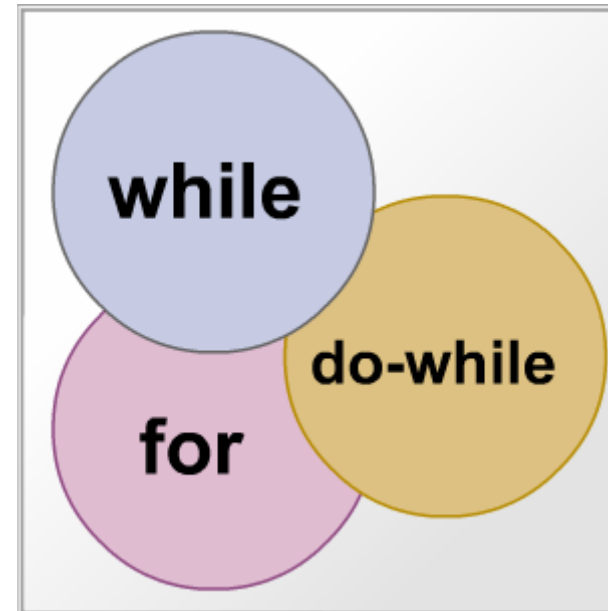
Switch case

- The switch – case statement can be used in place of **if-else-if** statement.
- It is used in situations where the expression being evaluated results in multiple values.

```
switch (expression){  
    case value1:  
        //statement;  
        break;  
    case value1:  
        //statement;  
        break;  
    ...  
    case valueN:  
        //statement;  
        break;  
    default:  
        //default statement;  
}
```

Introduction to Loops

- A loop comprises a statement or a block of statements that are executed repeatedly until a particular condition evaluates to true or false
- Loop statements in Java are:
 - While
 - Do while
 - For



while loop

- Used to execute a statement or a block of statements while a particular condition is true. The condition is checked before statements are executed.

```
while (condition)
{
    //one or more statements;
}
```

do while

- Check condition at the end of the loop rather than the beginning to ensure that the loop is executed at least once.

```
do{  
    //one or more statements  
} while (condition);
```

For statement

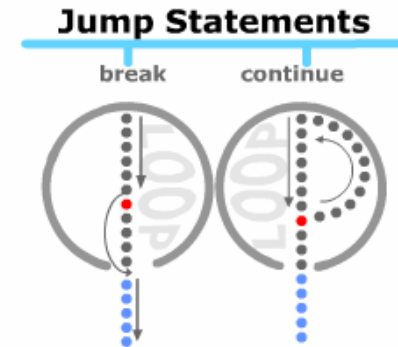
- The `for` loop provides a compact format for incorporating these features.

```
for (initialization; condition; increment / decrement )  
{  
    // one or more statements;  
}
```

Expand : nested loop.

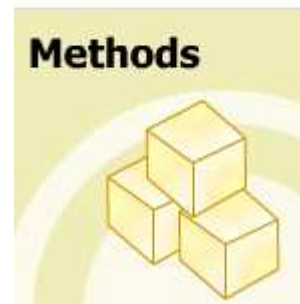
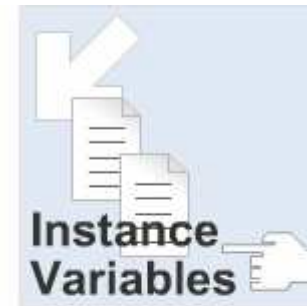
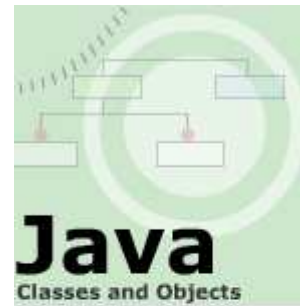
Jump statements

- Unconditionally transfer control to locations within program known as target of jump statements.
- **Break** :
 - terminates a statement sequence in a `switch` statement.
 - exits a loop.
 - is another form of `goto`
- **Continue** skips statements within a loop to proceed the next iteration of the loop



Module 4: Introducing class

- Creating class and object
- Instance variable
- Methods
- initialisers



Creating class

- A class defines a new data type.
- Declare a class

```
class <classname>
{
    ...
}
```

- Rules :

- Class name should be a noun.
- Class name can be in mixed case, with the first letter of each internal word capitalized.
- Class name should be simple, descriptive and meaningful.
- Class names cannot be Java keywords.

Class names cannot begin with a digit; however they can begin with a dollar(\$) symbol or an underscore character.

Constructor

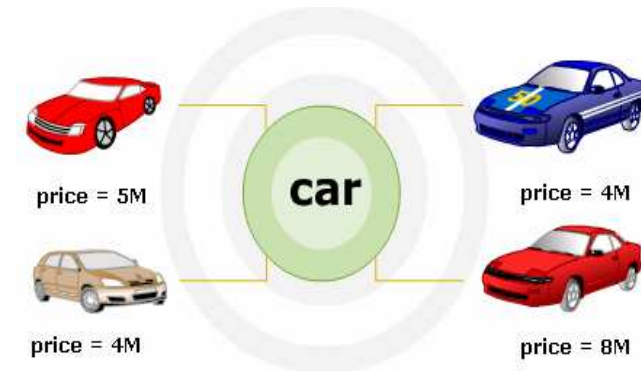
- Special methods are used to initialize member variables of the class.
- It has the same name as the Class name and does not have a return type.
- Called automatically and immediately after an object is created.
- Two types of constructors:
 - Parameterized constructors
 - Implicit or default constructors

Creating object

- Objects are declared to represent the class.
- Obtaining objects of a class is a two-step process. They are:
 - First, a variable of the class type has to be declared. The variable does not define an object. It is a variable that can refer to an object.
 - Second, an actual physical copy of the object must be acquired and assigned to that variable. It is done by using the `new` operator.
- The **new** operator dynamically allocates memory for an object and returns a reference to it.
- All class objects must be dynamically allocated

Instance variables

- Used to store information about an entity



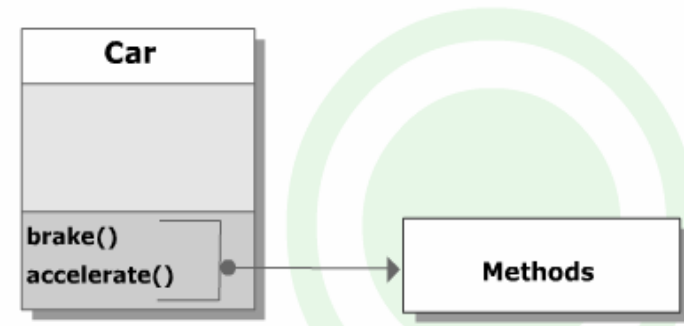
The benefit of using instance variables over local variables is that you declare only one instance variable and use it for all instances of the class. The name of instance variable is shared across all instances, but each instance has its own copy of instance variable.

Method

- A method is defined as the actual implementation of an operation on an object.

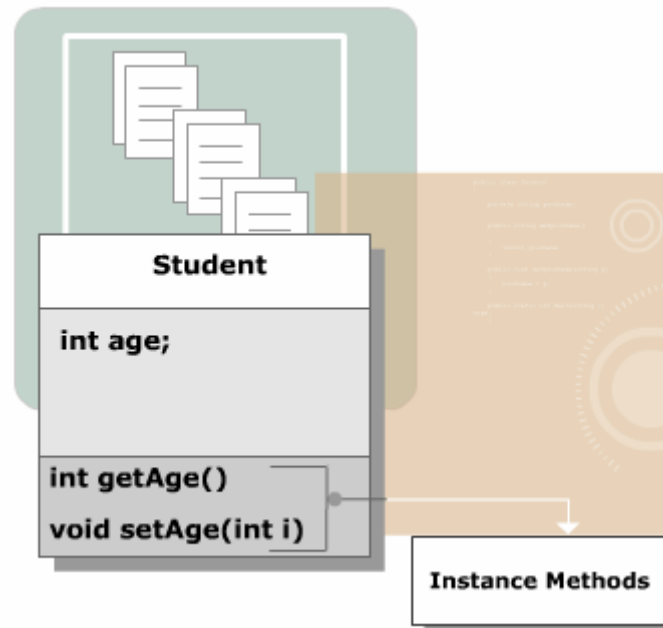
- **Syntax :**

```
access_specifier modifier datatype  
    method_name (parameter_list)  
{  
    //body of the method  
}
```



Instance Method

- Invoked by an instance object and can access instance variables.



Passing arguments to method

- **Pass by value:**
 - Primitive data type, change is local to method
- **Pass by reference:**
 - Change the value of parameters
 - Reference data type parameters are passed by value and not by reference, ie. when the method returns, the passed-in reference still references the same object as before.

VARIABLE ARGUMENT METHODS

- Variable argument allows calling a method with variable number of arguments. This is used when the number of arguments to be passed to a method is not known at the time of writing of a method.

```
datatype method_name (type ... variable)
{
    //method body
}
```

Initializers

- Are small pieces of code embedded in curly braces that perform initialization.

class field initializer



The class field initializer initializes class variables or class fields. It evaluates an expression with an assignment operator and assigns the result to a class field before executing any of its methods.

class block initializer



The class block initializer initializes complex classes. It consists of the `static` keyword, an open and close brace, and the initialization code.

object block initializer



The object block initializer initializes complex objects. It consists of an open and close brace and the initialization code.

object field initializer



The object field initializer initializes instance variables or object fields. It evaluates an expression with an assignment operator and assigns the result to an object field when an object is created and the constructor is called.

SUMMARY

- **Module 3:**
 - Decision – Making Statements
 - Introduction to Loops
 - Jump Statements
- **Module 4:**
 - Creating class and object
 - Instance variable
 - Methods
 - initialisers