3.3 The **switch** STATEMENT

```
switch([redacted]_Expression)
{
 case Case_Label:
        statements
        …
        break;
    case Case_Label:
        statements
        …
        break;

    default:
        statements
         …
       break;
}
```

- Another way to program multibranch selection
- Uses [redacted]_Expression to decide which way to branch
- [redacted]g_Expression must be *char, int, short or byte*.
- *Controlling Expression* and *Case_Label* must be same type.

Multibranch selection: **switch**

```
switch(Controlling_Expression)
{
 case Case_Label:
      statements
      …
      break;
   case Case_Label:
      statements
      …
      break;

   default:
      statements
       …
     break;
}
```

- When a `break` statement is encountered, control goes to the first statement after the `switch`.
- `break` may be omitted
- Can have any number of cases
- `default` case is optional

The `switch` Statement

- The `switch` statement
  - » a mutltiway branch
  - » makes a decision based on an *integral* (integer or character) expression.
  - » begins with the keyword `switch` followed by an integral expression in parentheses and called the [redacted].

The `switch` Statement, cont.

- A list of cases follows, enclosed in braces.
- Each case consists of the keyword `case` followed by
  - » a constant called [redacted]
  - » a colon
  - » a list of statements.
- The list is searched for a case label matching the controlling expression.

The `switch` Statement, cont.

- The action associated with a matching case label is executed.
- If no match is found, the case labeled [redacted] is executed.
  - » The `default` case is [redacted], but recommended, even if it simply prints a message.
- Repeated case labels are not allowed.

`switch` Example

```
switch(seatLocationCode)
{
 case 1:
    System.out.println("Orchestra");
    price = 40.00;
    break;
 case 2:
    System.out.println("Mezzanine");
    price = 30.00;
    break;
 case 3:
    System.out.println("Balcony");
    price = 15.00;
    break;
 default:
    System.out.println("Unknown seat code");
    break;
}
```

Output if `seatLocationCode` is 2:

Listing 3.5

- Listing 3.5 A Switch Statement
  - » MultipleBirths.java

```
// Listing 3.5 A Switch Statememts
import java.util.Scanner;

public class MultipleBirths
{
   public static void main(String[] args)
   {
      int numberOfBabies;
      System.out.print("Enter number of babies: ");
      Scanner keyboard = new Scanner(System.in);
      numberOfBabies = keyboard.nextInt();
```

```
switch (numberOfBabies)
      {
         case 1:
            System.out.println("Congratulations.");
            break;
         case 2:
            System.out.println("Wow. Twins.");
            break;
         case 3:
            System.out.println("Wow. Triplets.");
            break;
         case 4:
         case 5:
            System.out.println("Unbelieveable.");
            System.out.println(numberOfBabies + " babies");
            break;
         default:
            System.out.println("I don't believe you.");
            break;
      }
   }
}
```

4
5
6

```
C:₩WINDOWS₩system32₩cmd.exe
Enter number of babies: 4
Unbelieveable.
4 babies
계속하려면 아무 키나 누르십시오 . . .
```

```
C:₩WINDOWS₩system32₩cmd.exe
Enter number of babies: 5
Unbelieveable.
5 babies
계속하려면 아무 키나 누르십시오 . . .
```

```
C:₩WINDOWS₩system32₩cmd.exe
Enter number of babies: 6
I don't believe you.
계속하려면 아무 키나 누르십시오 . . .
```

Enumerations

- Consider a need to [redacted] contents of a variable to certain values
- An enumeration lists the values a variable can have
- Example

```
enum MovieRating {E, A, B}
MovieRating rating;
rating = MovieRating.A;
```

Enumerations

- Now possible to use in a **`switch`** statement

```
switch (rating)
{
   case E: //Excellent
      System.out.println("You must see this movie!");
      break;
   case A: //Average
      System.out.println("This movie is OK, but not great.");
      break;
   case B: // Bad
      System.out.println("Skip it!");
      break;
   default:
      System.out.println("Something is wrong.");
}
```

```
public class EnumTest
{
   enum MovieRating {E, A, B}
   public static void main(String[] args)
   {

      MovieRating rating;
      rating = MovieRating.A;
      switch (rating)
      {
                  case E: //Excellent
                        System.out.println("You must see this movie!");
                        break;
                  case A: //Average
                        System.out.println("THis movie is OK, but not great.");
                        break;
                  case B: //Bad
                        System.out.println("Skip it!");
                        break;
                  default:
                        System.out.println("SOmething is wrong");
      }
   }
}
```

EnumTest$1	2019-09-22 오후 ...	CLASS 파일	1KB
EnumTest$MovieRating	2019-09-22 오후 ...	CLASS 파일	1KB
EnumTest	2019-09-22 오후 ...	CLASS 파일	1KB
EnumTest	2019-09-22 오후 ...	JAVA 파일	1KB

C:₩WINDOWS₩system32₩cmd.exe

```
THis movie is OK, but not great.
계속하려면 아무 키나 누르십시오 . . .
```

Enumerations

- An even better choice of descriptive identifiers for the constants

```
enum MovieRating
      {EXCELLENT, AVERAGE, BAD}
rating = MovieRating.AVERAGE;

case EXCELLENT:  ...
```

끝