## Class Declaration

```
public class CashRegister
{
   private int itemCount;
   private double totalPrice;

   public void addItem(double price)
   {
      itemCount++;
      totalPrice = totalPrice + price;
   }
   . . .
}
```

Instance variables

Method

## Selected Operators and Their Precedence

*(See Appendix B for the complete list.)*

| | |
|---|---|
| `[]` | Array element access |
| `++ -- !` | Increment, decrement, Boolean *not* |
| `* / %` | Multiplication, division, remainder |
| `+ -` | Addition, subtraction |
| `< <= > >=` | Comparisons |
| `== !=` | Equal, not equal |
| `&&` | Boolean *and* |
| `\|\|` | Boolean *or* |
| `=` | Assignment |

## Conditional Statement

```
if (floor >= 13)
{
   actualFloor = floor - 1;
}
else if (floor >= 0)
{
   actualFloor = floor;
}
else
{
   System.out.println("Floor negative");
}
```

Condition

Executed when condition is true

Second condition (optional)

Executed when all conditions are false (optional)

## Variable and Constant Declarations

```
int cansPerPack = 6;

final double CAN_VOLUME = 0.335;
```

Type, Name, Initial value

## Method Declaration

```
public static double cubeVolume(double sideLength)
{
   double volume = sideLength * sideLength * sideLength;
   return volume;
}
```

Modifiers, Return type, Parameter type and name

Exits method and returns result.

## Mathematical Operations

| | |
|---|---|
| `Math.pow(x, y)` | Raising to a power $x^y$ |
| `Math.sqrt(x)` | Square root $\sqrt{x}$ |
| `Math.log10(x)` | Decimal log $\log_{10}(x)$ |
| `Math.abs(x)` | Absolute value $\lvert x \rvert$ |
| `Math.sin(x)`, `Math.cos(x)`, `Math.tan(x)` | Sine, cosine, tangent of $x$ ($x$ in radians) |

## String Operations

```
String s = "Hello";
int n = s.length(); // 5
char ch = s.charAt(1); // 'e'
String t = s.substring(1, 4); // "ell"
String u = s.toUpperCase(); // "HELLO"
if (u.equals("HELLO")) ... // Use equals, not ==
for (int i = 0; i < s.length(); i++)
{
   char ch = s.charAt(i);
   Process ch
}
```

## Loop Statements

```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

Condition

Executed while condition is true

```
for (int i = 0; i < 10; i++)
{
   System.out.println(i);
}
```

Initialization, Condition, Update

```
do
{
   System.out.print("Enter a positive integer: ");
   input = in.nextInt();
}
while (input <= 0);
```

Loop body executed at least once

```
for (double value : values)
{
   sum = sum + value;
}
```

Set to a new element in each iteration

An array or collection

Executed for each element

## Input

```
Scanner in = new Scanner(System.in);
  // Can also use new Scanner(new File("input.txt"));

int n = in.nextInt();
double x = in.nextDouble();
String word = in.next();
String line = in.nextLine();

while (in.hasNextDouble())
{
   double x = in.nextDouble();
   Process x
}
```

## Output

Does not advance to new line.

```
System.out.print("Enter a value: ");
```

Use + to concatenate values.

```
System.out.println("Volume: " + volume);
```

Field width, Precision

```
System.out.printf("%-10s %10d %10.2f", name, qty, price);
```

Left-justified string, Integer, Floating-point number

```
PrintWriter out = new PrintWriter("output.txt");

out.close();
```

Use print/println/printf to write output to file.

Remember to close output file.

## Arrays

Element type, Element type, Length

```
int[] numbers = new int[5];
int[] squares = { 0, 1, 4, 9, 16 };
int[][] magicSquare =
   {
      { 16,  3,  2, 13},
      {  5, 10, 11,  8},
      {  9,  6,  7, 12},
      {  4, 15, 14,  1}
   };

for (int i = 0; i < numbers.length; i++)
{
   numbers[i] = i * i;
}

for (int element : numbers)
{
   Process element
}

System.out.println(Arrays.toString(numbers));
   // Prints [0, 1, 4, 9, 16]
```

All elements are zero.

## Array Lists

Initially empty, Use wrapper type, Integer, Double, etc., for primitive types.

Element type

```
ArrayList<String> names = new ArrayList<String>();
```

Add elements to the end

```
names.add("Ann");
names.add("Cindy"); // [Ann, Cindy], names.size() is now 2

names.add(1, "Bob"); // [Ann, Bob, Cindy]
names.remove(2); // [Ann, Bob]
names.set(1, "Bill"); // [Ann, Bill]

String name = names.get(0); // Gets "Ann"
System.out.println(names); // Prints [Ann, Bill]
```

## Linked Lists, Sets, and Iterators

```
LinkedList<String> names = new LinkedList<String>();
names.add("Bob"); // Adds at end

ListIterator<String> iter = names.listIterator();
iter.add("Ann"); // Adds before current position

String name = iter.next(); // Returns "Ann"
iter.remove(); // Removes "Ann"

Set<String> names = new HashSet<String>();
names.add("Ann"); // Adds to set if not present
names.remove("Bob"); // Removes if present

Iterator<String> iter = names.iterator();
while (iter.hasNext())
{
   Process iter.next()
}
```

## Maps

Key type, Value type

```
Map<String, Integer> scores = new HashMap<String, Integer>();

scores.put("Bob", 10);
Integer score = scores.get("Bob");
```

Returns null if key not present

```
for (String key : scores.keySet())
{
   Process key and scores.get(key)
}
```