Ticket Machine Project(s)

Understanding the basic contents of classes

Produced by: Dr. Siobhán Drohan

(based on Chapter 2, Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling)

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Topic List

- Data types:
  - primitive
  - objects
- Demo of naïve ticket machine
- Inside classes:
  - fields
  - constructors
  - methods:
    - accessors
    - mutators
  - assignment statements
- Demo of better ticket machine
- Making choices: conditional statements (if)

Data Types

- Java uses two kinds of types:
  - Primitive types
  - Object types

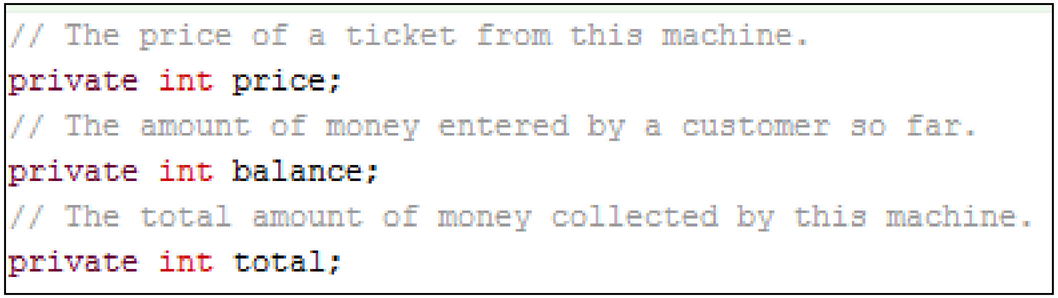
- A field's data type determines the values it may contain, plus the operations that may be performed on it.

Primitive Data Types

- Java programming language supports eight primitive data types.

- A primitive type is predefined by the language and is named by a reserved keyword.

- A primitive type is highlighted red when it is typed into BlueJ e.g.



```
// The price of a ticket from this machine.
private int price;
// The amount of money entered by a customer so far.
private int balance;
// The total amount of money collected by this machine.
private int total;
```

Primitive Data Types (for whole numbers)

Type	Byte-size	Minimum value (inclusive)	Maximum value (inclusive)	Typical Use
byte	8-bit	-128	127	Useful in applications where memory savings apply.
short	16-bit	-32,768	32,767	
int	32-bit	-2,147,483,648	2,147,483,647	Default choice.
long	64-bit	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	Used when you need a data type with a range of values larger than that provided by int.

Primitive Data Types (for decimal numbers)

Type	Byte-size	Minimum value (inclusive)	Maximum value (inclusive)	Typical Use
float	32-bit	*Beyond the scope of this lecture .* *There is also a loss of precision in this data-type that we will cover in later lectures.*		Useful in applications where memory savings apply.
double	64-bit			Default choice.

Primitive Data Types (others)

Type	Byte-size	Minimum value (inclusive)	Maximum value (inclusive)	Typical Use
char	16-bit	'\u0000' (or 0)	'\uffff' (or 65,535).	Represents a Unicode character.
boolean	1-bit	n/a		Holds either true or false and is typically used as a flag.

http://en.wikipedia.org/wiki/List_of_Unicode_characters

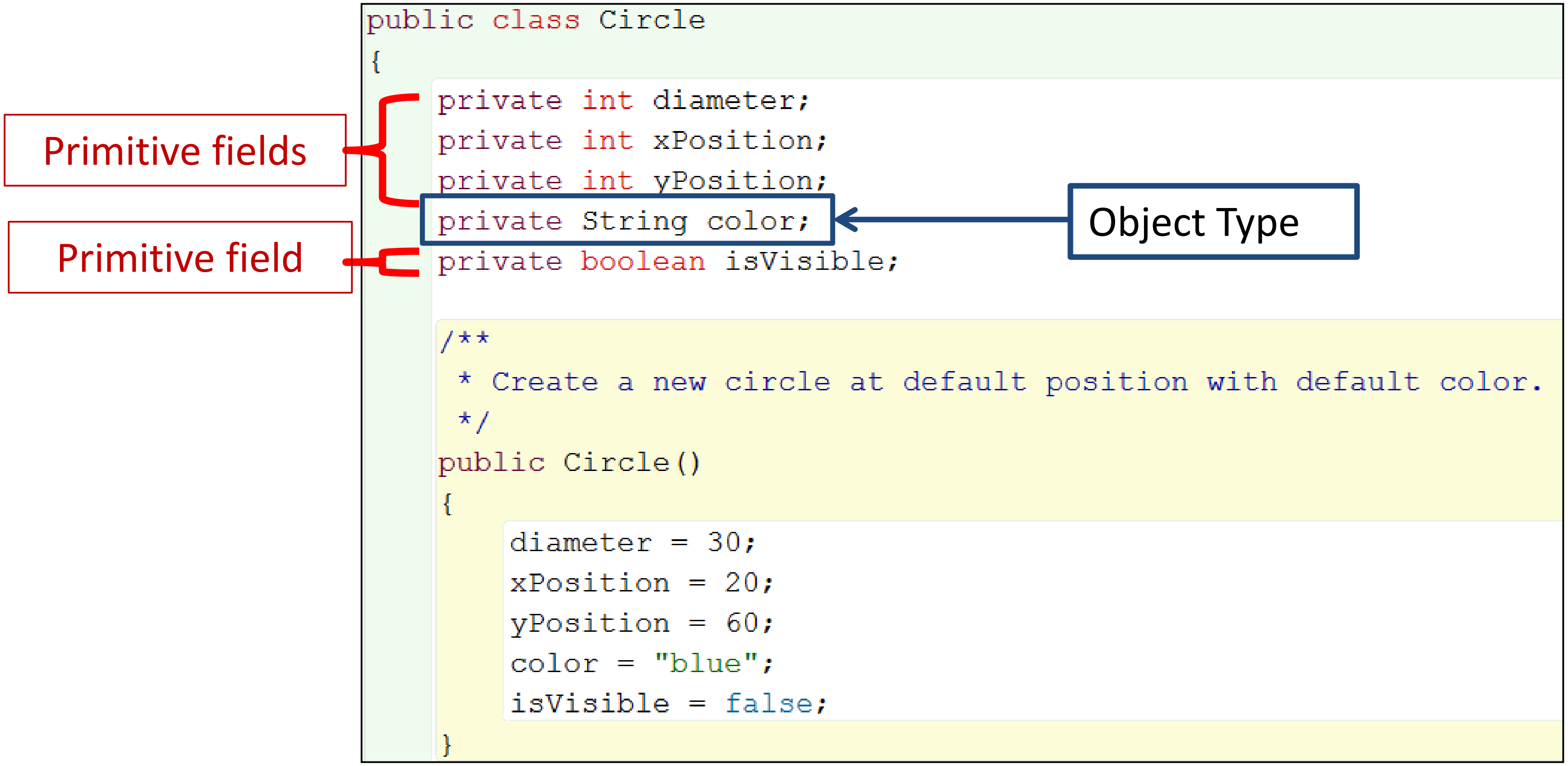
Default values

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Object Types

- All types that are not primitive are object types.



```
public class Circle
{
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new circle at default position with default color.
     */
    public Circle()
    {
        diameter = 30;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }
```

Primitive fields

Primitive field

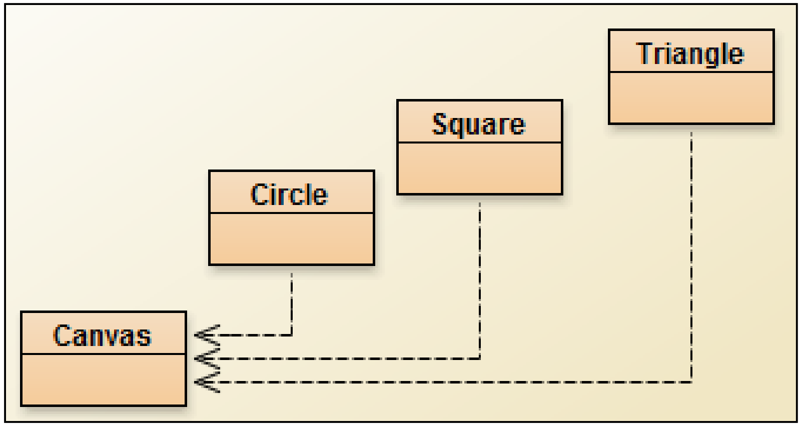
Object Type

Object Types

- Includes classes from standard java library e.g. String:

```
    private String color;
```

- Also includes user defined classes e.g. Square, Circle, etc.



Topic List

- Data types:
  - primitive
  - objects
- Demo of naïve ticket machine
- Inside classes:
  - fields
  - constructors
  - methods:
    - accessors
    - mutators
  - assignment statements
- Demo of better ticket machine
- Making choices: conditional statements (if)

Ticket machine – an external view

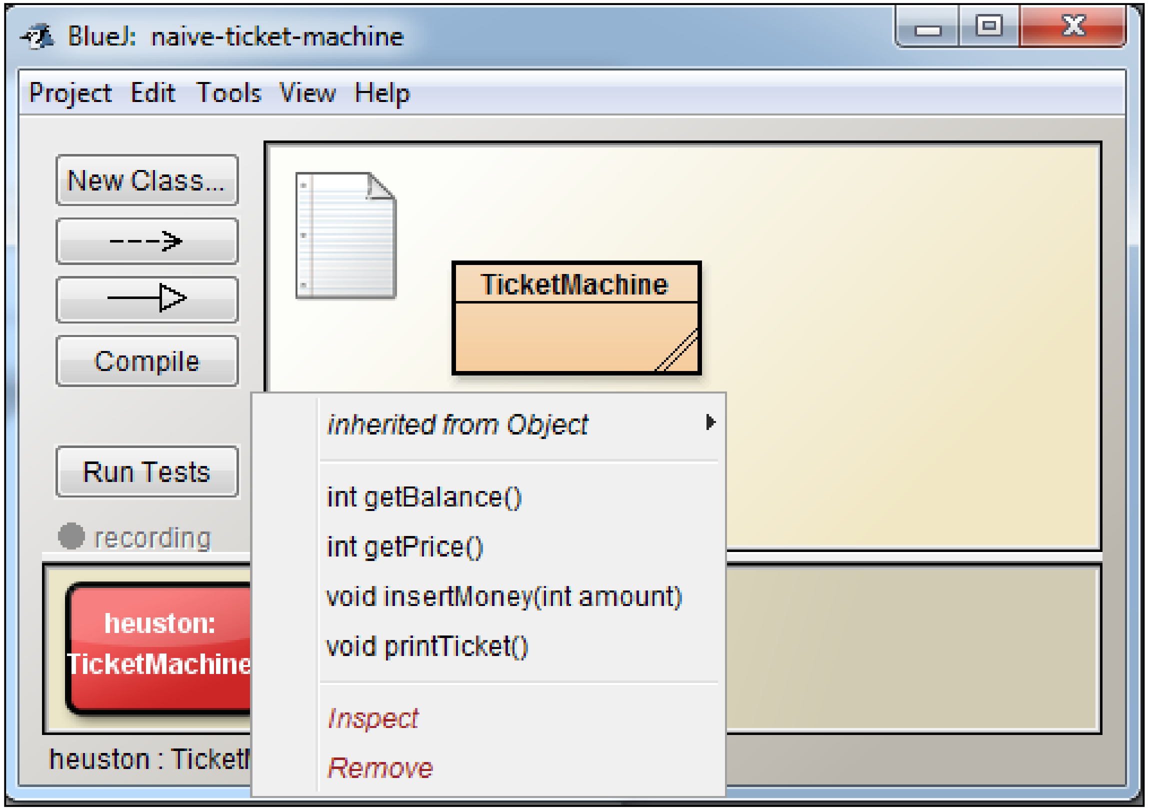
- Exploring the behavior of a typical ticket machine (e.g. the *naive-ticket-machine*):
  - Machines supply tickets of a fixed price.
    - How is that price determined?
  - How is 'money' entered into a machine?
  - How does a machine keep track of the money that is entered?

Demo

Exploring the behaviour of the naïve ticket machine

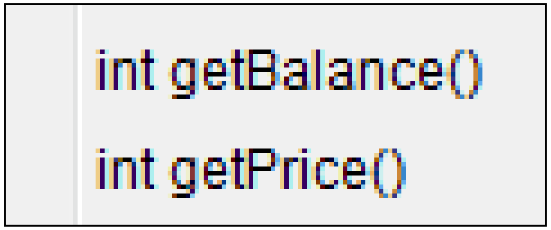
Ticket machines – an internal view

Interacting with an object gives us clues about its behavior.



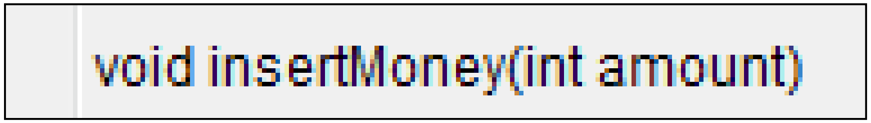
int getBalance()
int getPrice()
void insertMoney(int amount)
void printTicket()

Ticket machines – an internal view



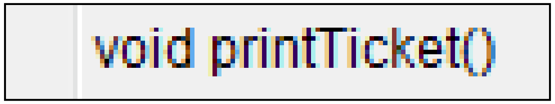
```
int getBalance()
int getPrice()
```

Returns a whole number (int) representing the balance or price of the ticket. Both methods have no parameters; they don’t need any information to do their task.



```
void insertMoney(int amount)
```

Allows the user to insert money (an int value parameter) into the ticket machine. Doesn’t return anything (it is void).



```
void printTicket()
```

Prints the ticket to the console window. Doesn’t return anything (it is void).

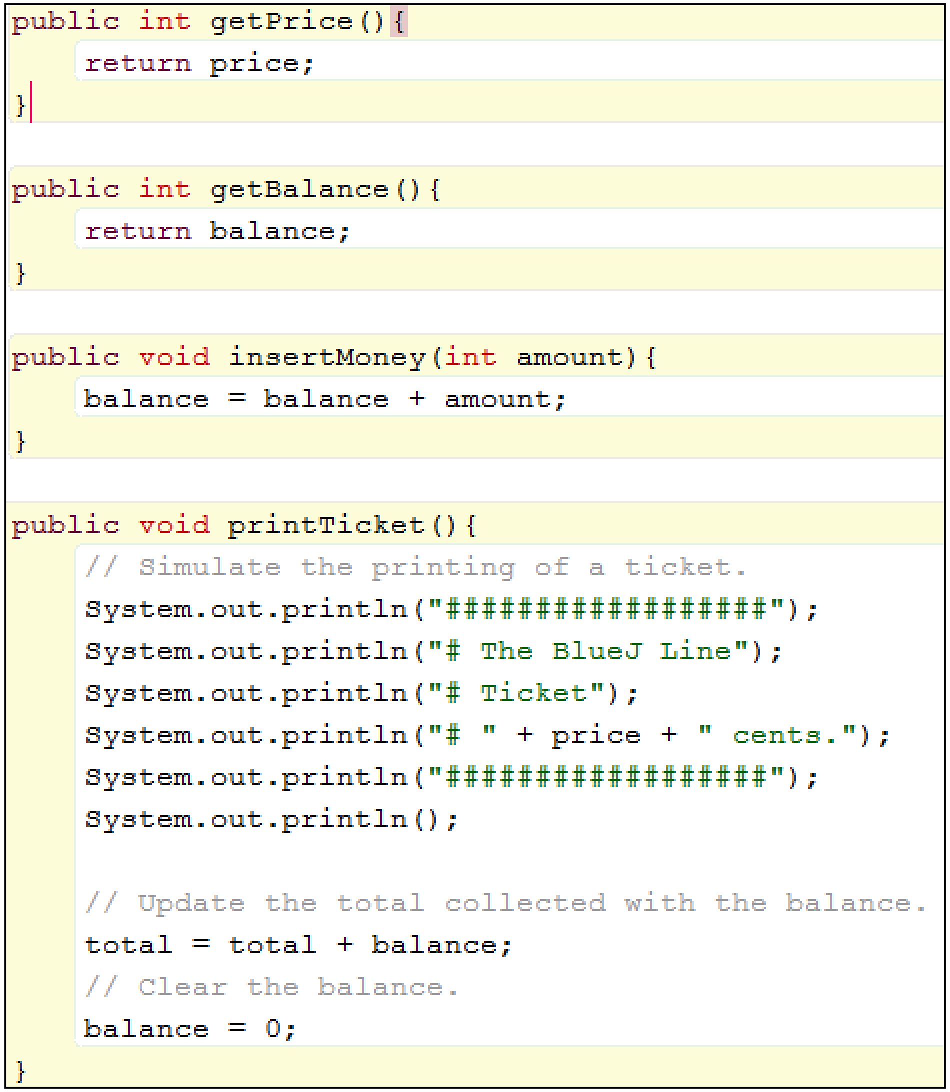
Topic List

- Data types:
  - primitive
  - objects
- Demo of naïve ticket machine
- Inside classes:
  - fields
  - constructors
  - methods:
    - accessors
    - mutators
  - assignment statements
- Demo of better ticket machine
- Making choices: conditional statements (if)

Ticket machines – an internal view

- Looking inside allows us to determine how that behavior is provided or implemented.

- All Java classes have a similar-looking internal view.



```java
public int getPrice(){
    return price;
}

public int getBalance(){
    return balance;
}

public void insertMoney(int amount){
    balance = balance + amount;
}

public void printTicket(){
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Basic class structure

```
public class TicketMachine
{
    //Inner part of the class omitted.
}
```

The outer wrapper of TicketMachine

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

The contents of a class

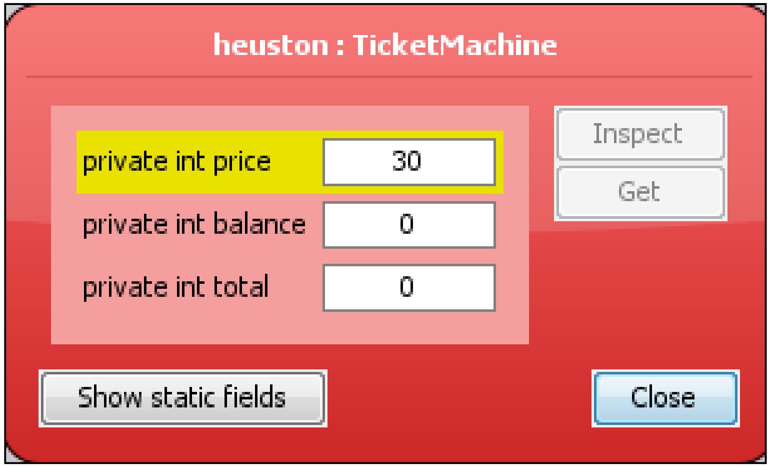
Instance fields

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

- Variables store values for an object.
- These variables are typically called instance fields / instance variables.

```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    //Further details omitted.
}
```

- Instance fields define the ***state*** of an object i.e. the values stored in the instance fields.



heuston : TicketMachine

private int price 30

private int balance 0

private int total 0

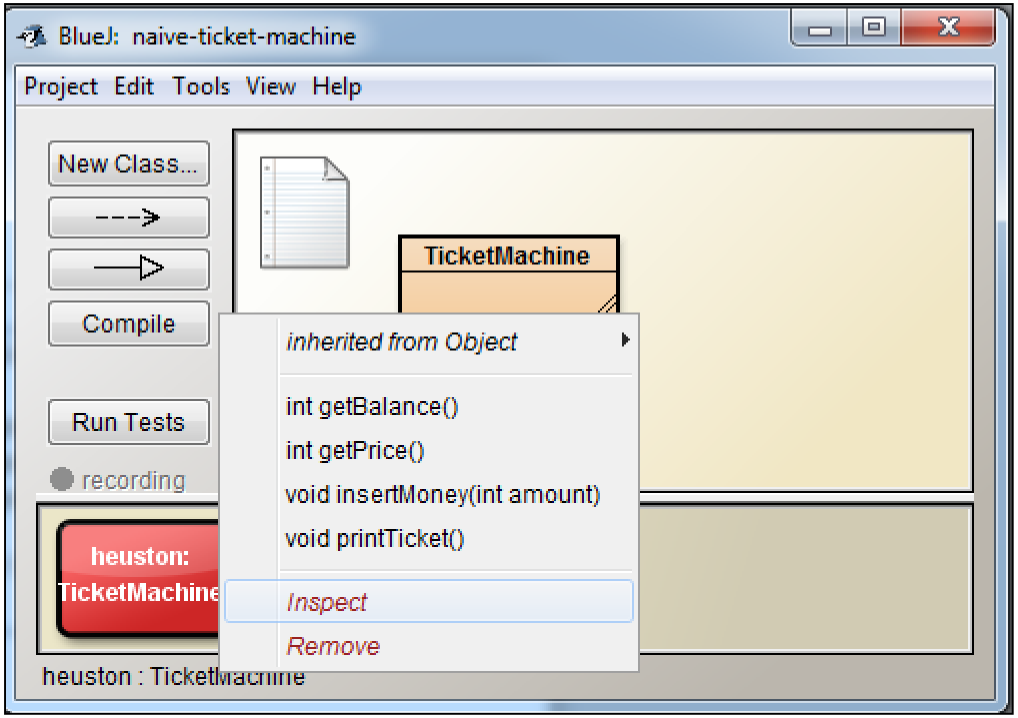
Instance fields

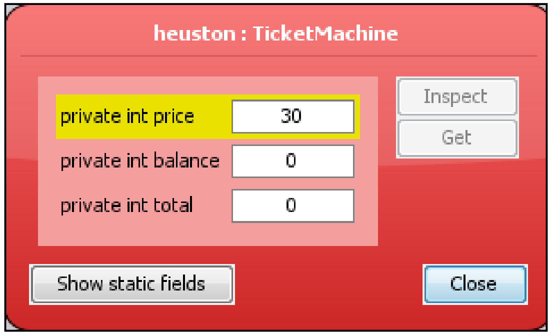
```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

In BlueJ, you can view the object state by either:

- right clicking on the object and selecting the *Inspect* option **OR**

- double clicking on the object.



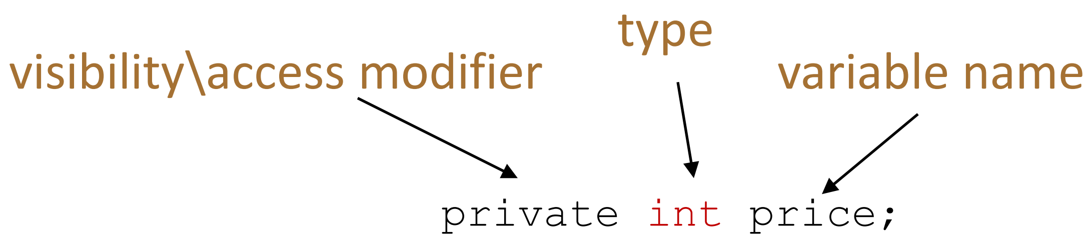

heuston : TicketMachine

private int price	30
private int balance	0
private int total	0

Instance fields

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    //Further details omitted.
}
```



visibility\access modifier

type

variable name

private int price;

Constructors

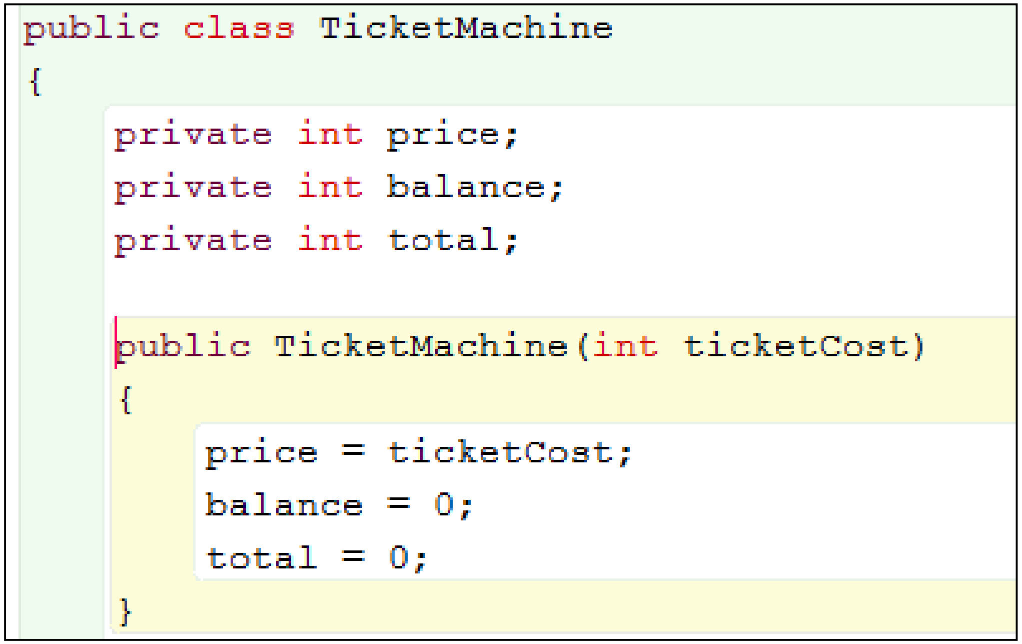
```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

- A constructor builds an object and initialises it to a starting state.

- They have the same name as their class.

- Their access modifier is public.

- They store initial values in the instance fields; they often receive external parameter values for this.



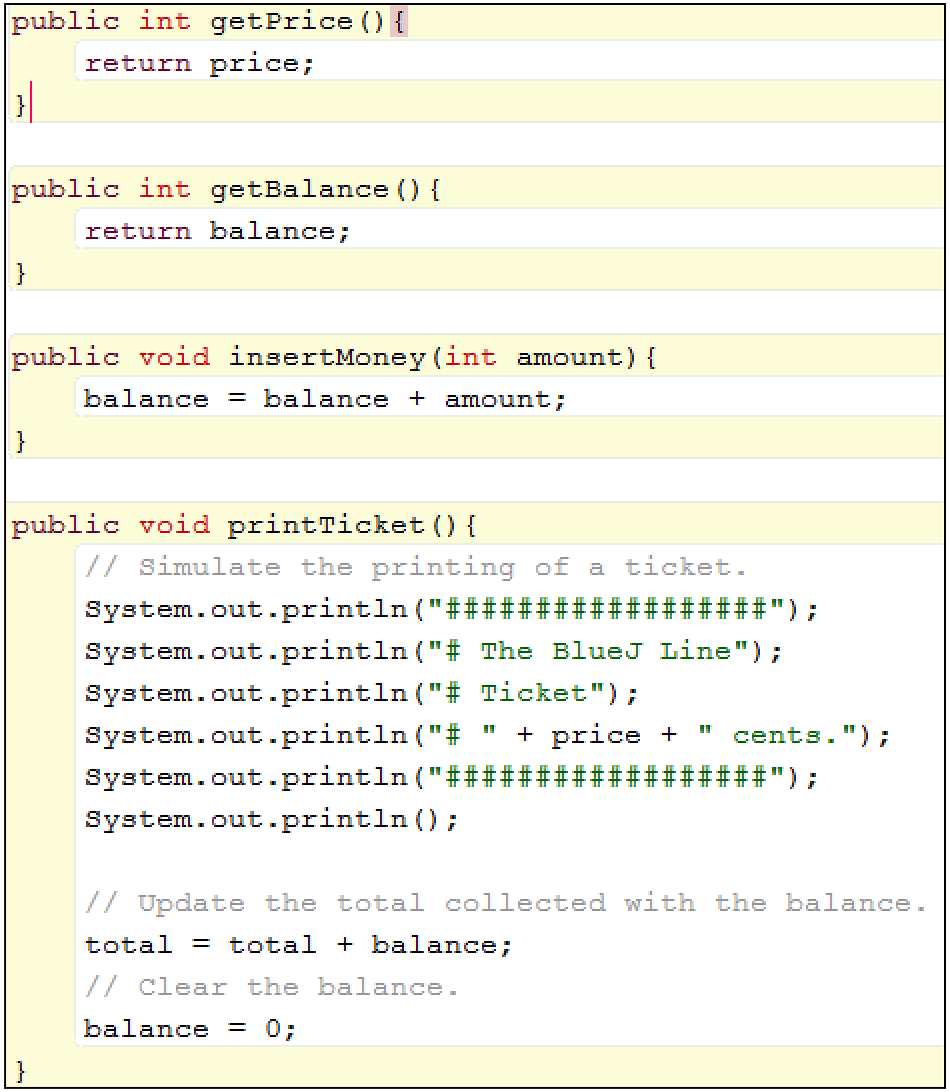
```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
        balance = 0;
        total = 0;
    }
```

Methods

```
public class ClassName
{
     //Instance Fields
     //Constructors
     //Methods
}
```

- Methods implement the behaviour of objects.

- Java uses methods to communicate with other classes.



```
public int getPrice(){
    return price;
}

public int getBalance(){
    return balance;
}

public void insertMoney(int amount){
    balance = balance + amount;
}

public void printTicket(){
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Method signature

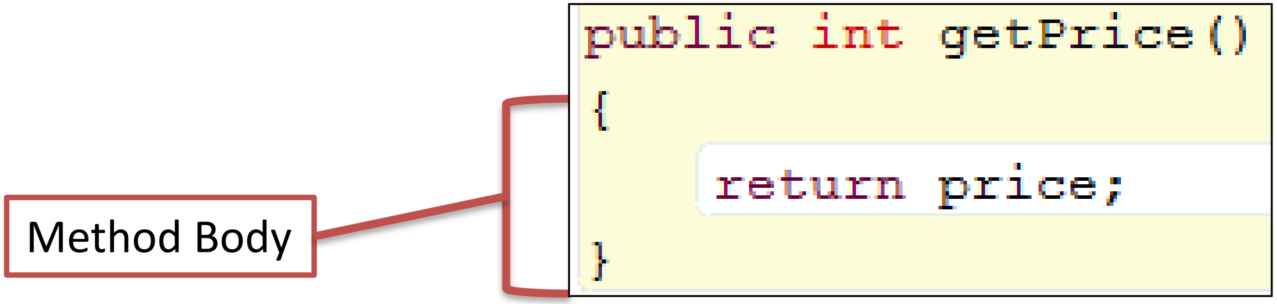
```
public class ClassName
{
     //Instance Fields
     //Constructors
     //Methods
}
```

The method signature consists of a method name and its parameter type list e.g.

getPrice()

insertMoney(int amount)

The method body encloses the method’s statements i.e. the code block for the method



```
public int getPrice()
{
      return price;
}
```

Method Body

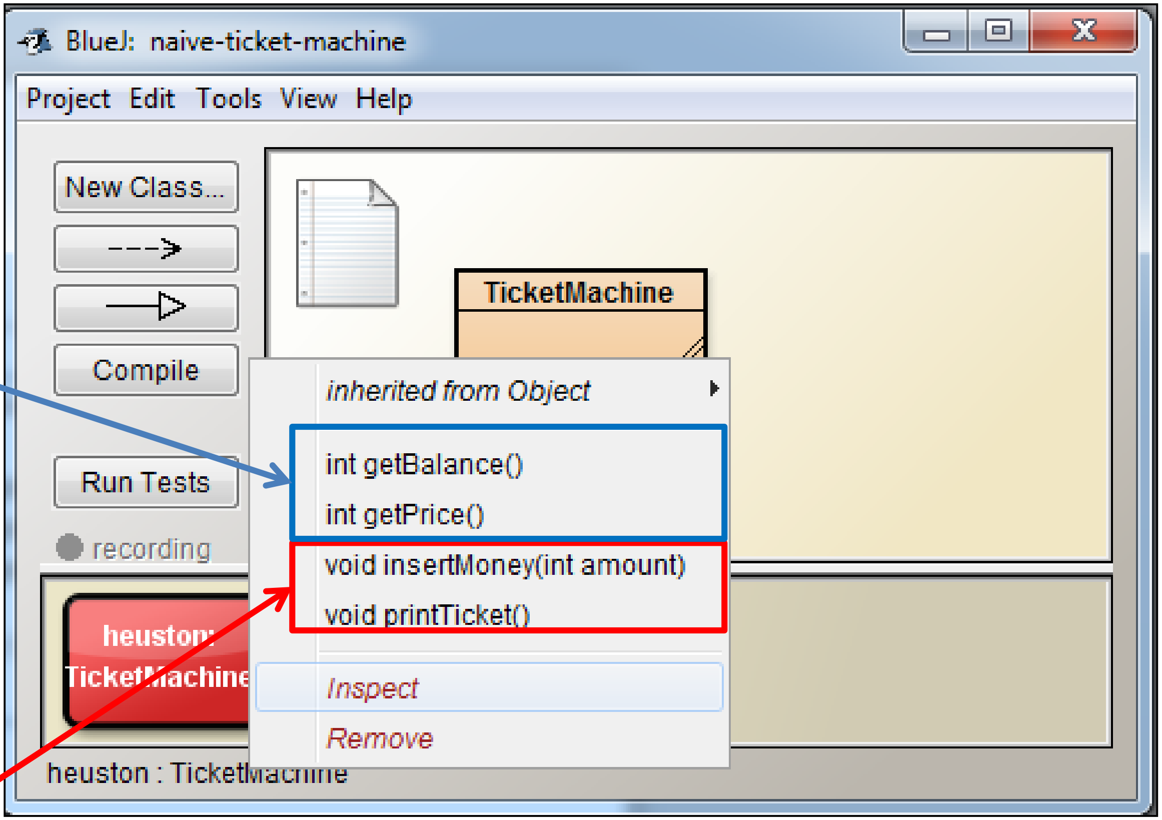
Method return types

```
public class ClassName
{
     //Instance Fields
     //Constructors
     //Methods
}
```

Methods can return information about an object via a return value.

The int before the method names mean that a whole number is returned from these methods. A method can only have one return type.

The void just before the method name means that nothing is returned from these methods.
void is a return type and must be included in the method signature if your method returns no information.

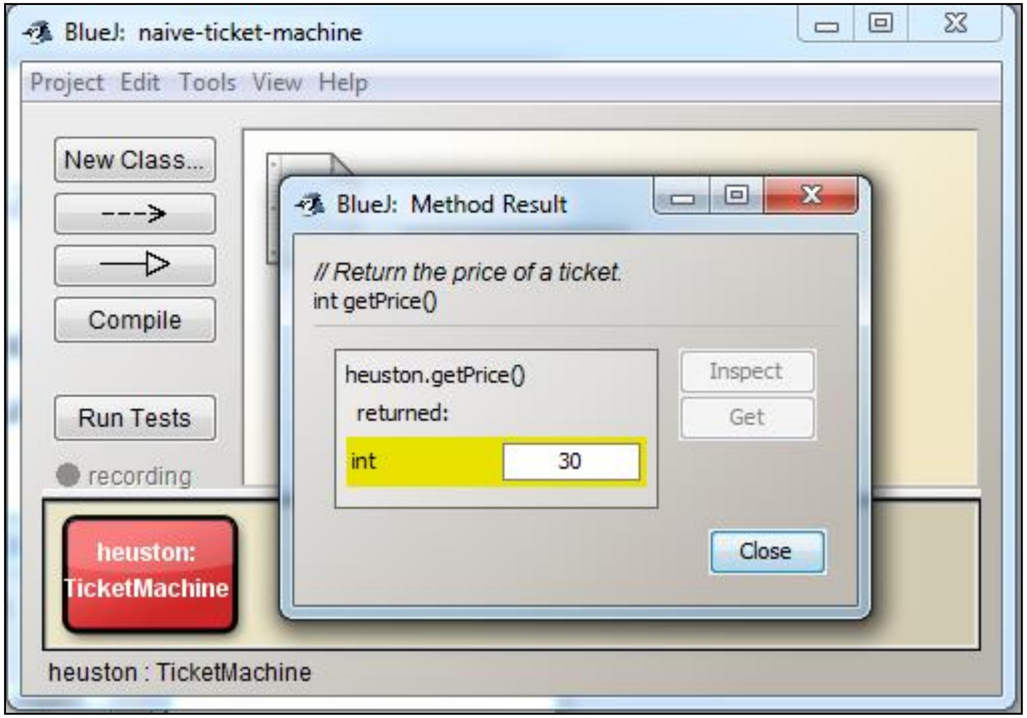


Return types

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

In BlueJ, when you call a method that returns data, a screen will pop up with the returned data e.g.

- the **getPrice()** method returns the whole number, 30.



BlueJ: naive-ticket-machine

BlueJ: Method Result

```
// Return the price of a ticket.
int getPrice()
```

heuston.getPrice()
returned:

int 30

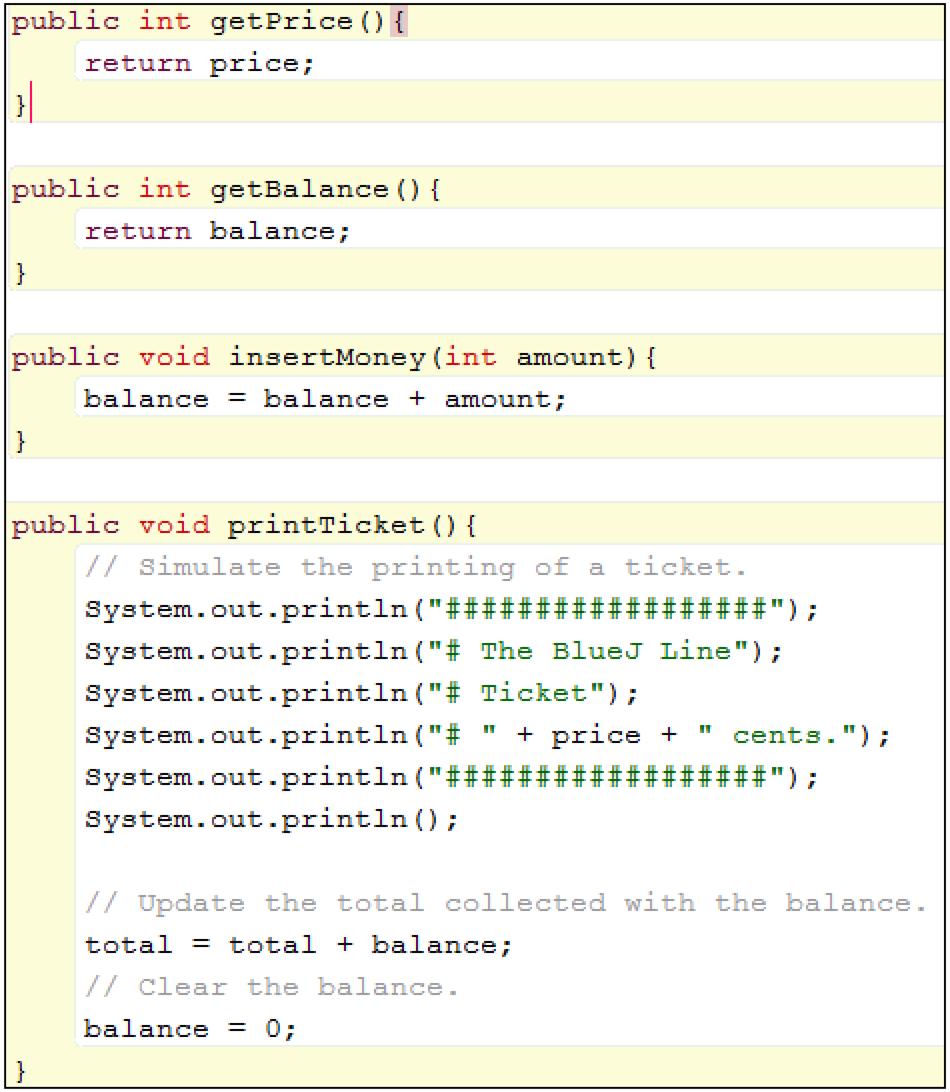
heuston : TicketMachine

Types of Methods

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

Now that we have covered method signature and return types, we are going to look at two specific “types” of methods i.e.

- Accessor methods
- Mutator methods



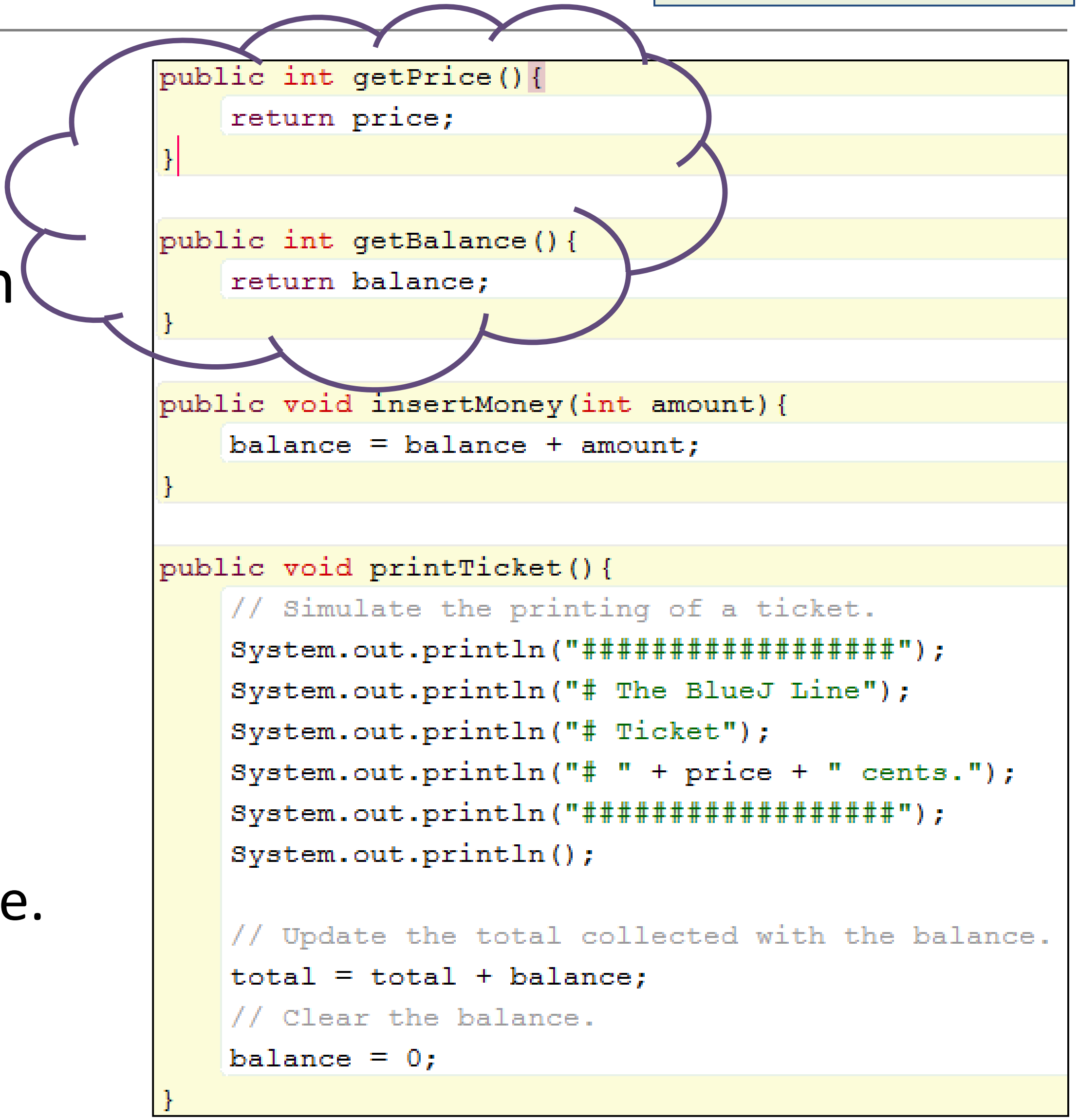
```
public int getPrice(){
    return price;
}

public int getBalance(){
    return balance;
}

public void insertMoney(int amount){
    balance = balance + amount;
}

public void printTicket(){
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Accessor methods

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

- Accessor methods return information about the state of an object.

- Typically they:
  - contain a return statement (as the last executable statement in the method).
  - define a return type.

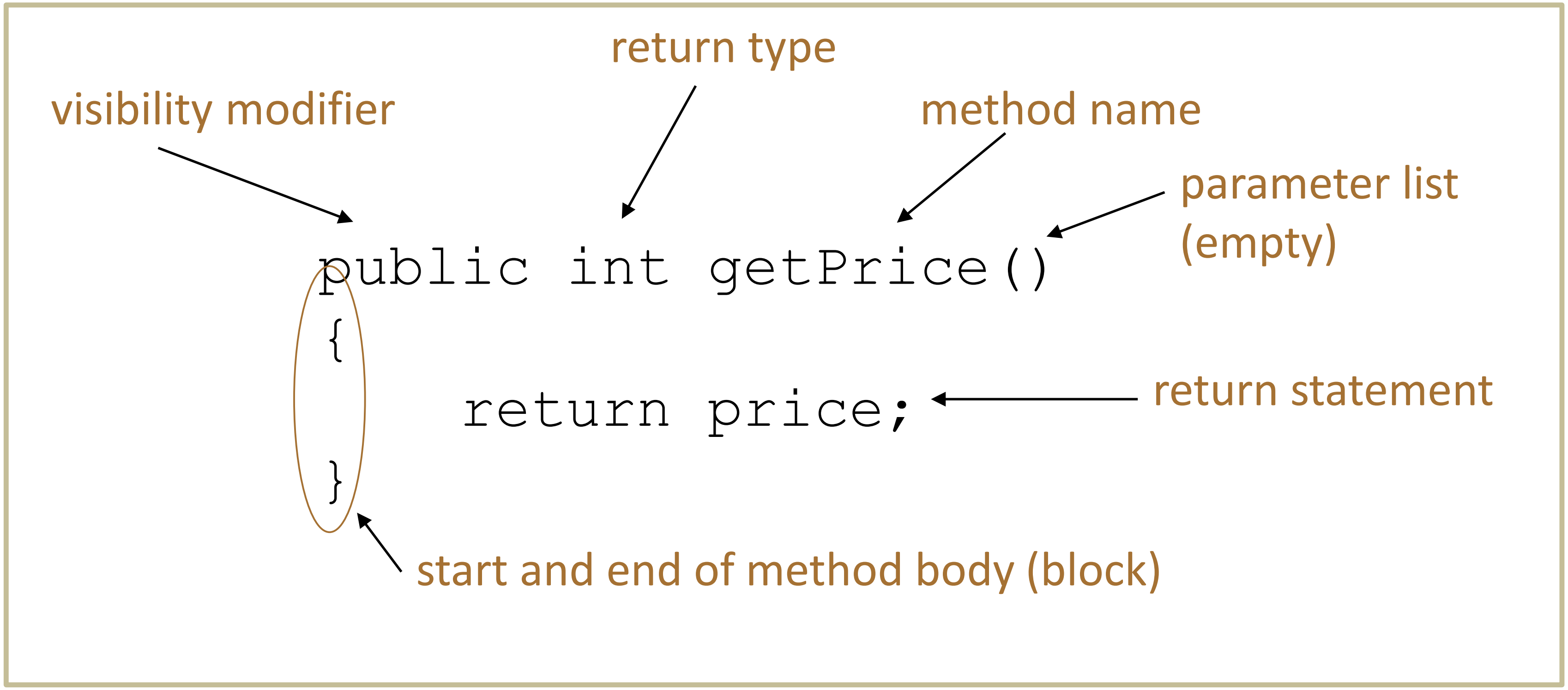


```
public int getPrice(){
    return price;
}

public int getBalance(){
    return balance;
}

public void insertMoney(int amount){
    balance = balance + amount;
}

public void printTicket(){
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Accessor/getter methods

```
public class ClassName
{
     //Instance Fields
     //Constructors
     //Methods
}
```

- ‘Getter’ methods are a specific type of accessor method.



```
public int getPrice()
{
        return price;
}
```

visibility modifier

return type

method name

parameter list (empty)

return statement

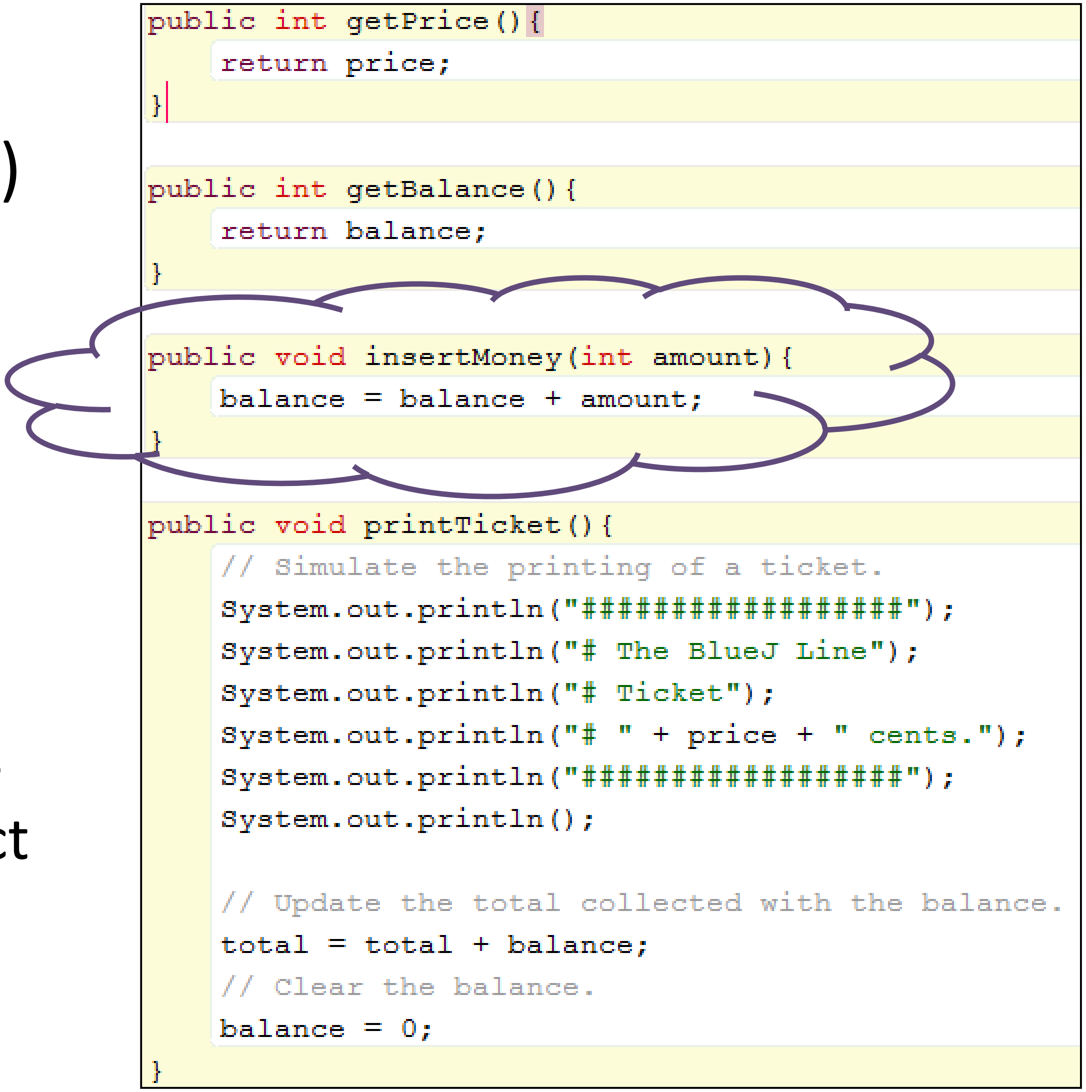
start and end of method body (block)

Mutator methods

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

- Mutator methods change (i.e. mutate!) an object’s state.

- Typically they:
  - contain an assignment statement
  - take in a parameter to change the object state.

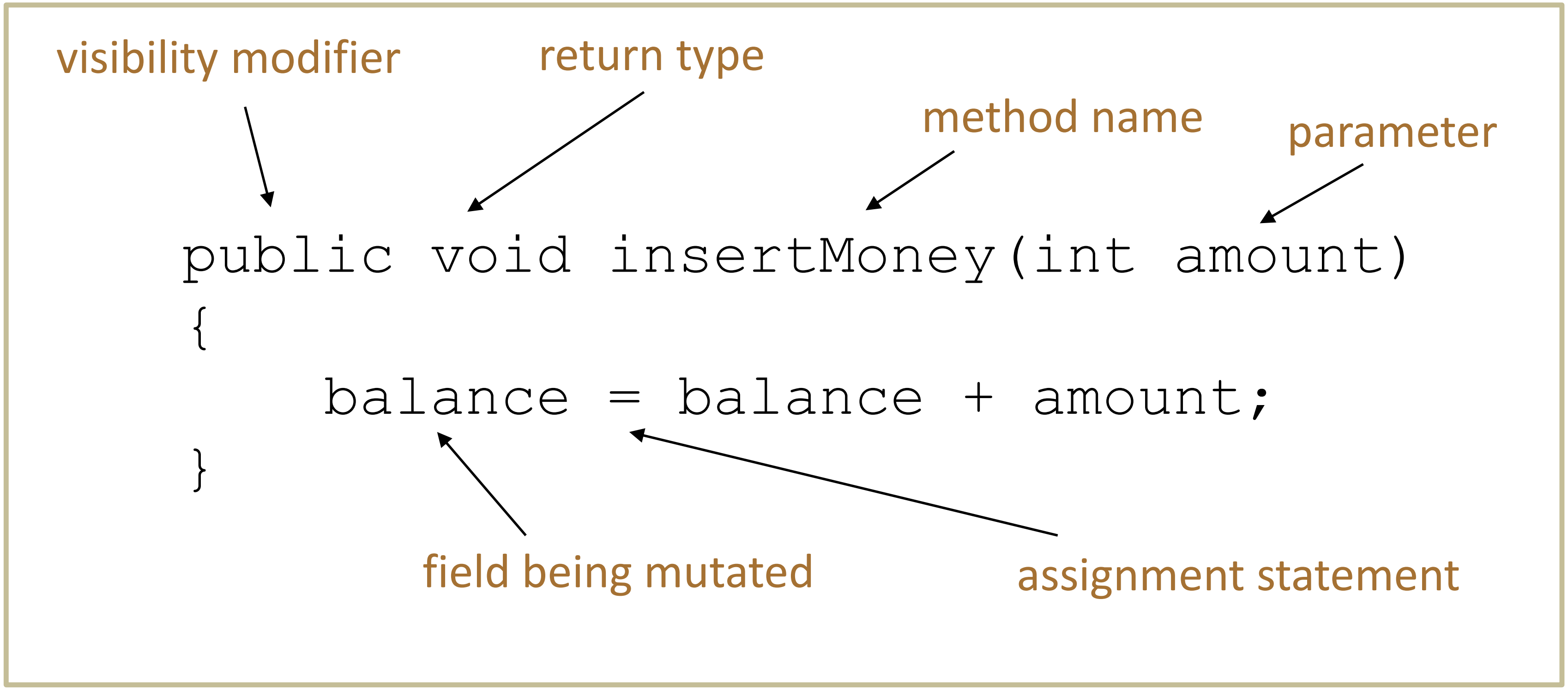


```
public int getPrice(){
    return price;
}

public int getBalance(){
    return balance;
}

public void insertMoney(int amount){
    balance = balance + amount;
}

public void printTicket(){
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Mutator/setter methods

```
public class ClassName
{
     //Instance Fields
     //Constructors
     //Methods
}
```

- ‘Setter’ methods are a specific type of mutator method.

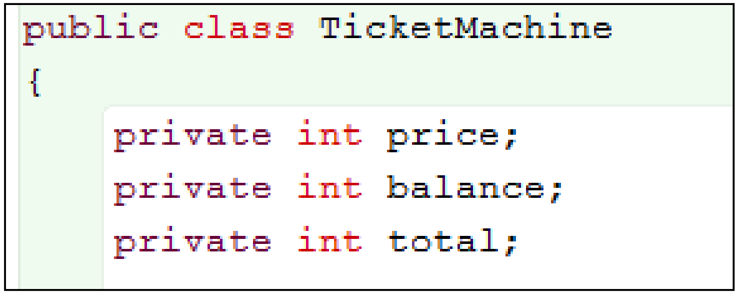


```
public void insertMoney(int amount)
{
    balance = balance + amount;
}
```

Getters/setters

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

- For each instance field in a class, you are normally asked to write:
  - A getter
  - A setter

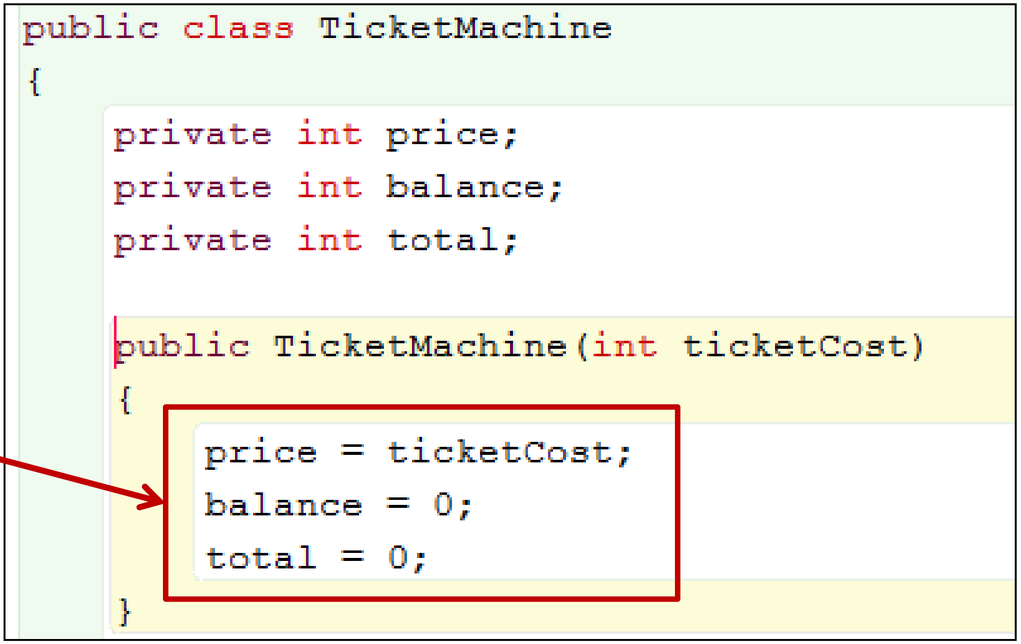


```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;
```

- However, depending on the design of your app, you may wish to not provide getters/setters for specific fields (more on this later!)

Assignment Statement

Values are stored in instance fields (and other variables) via assignment statements.



```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
        balance = 0;
        total = 0;
    }
```

Assignment Statement

Syntax	`variable = expression;`
Example	`price = ticketCost;`

- A variable stores a single value, so any previous value is lost.
- Assignment statements work by taking the value of what appears on the right-hand side of the operator and copying that value into a variable on the left-hand side.

Topic List

- Data types:
  - primitive
  - objects
- Demo of naïve ticket machine
- Inside classes:
  - fields
  - constructors
  - methods:
    - accessors
    - mutators
  - assignment statements
- Demo of better ticket machine
- Making choices: conditional statements (if)

Reflecting on the naïve ticket machine

- The behavior is inadequate in several ways:
  - No checks on the amounts entered.
  - No refunds.
  - No checks for a sensible initialisation.

- How can we do better?
  - We need more sophisticated behavior.

demo

Briefly explore the more sophisticated behaviour of the:

better ticket machine

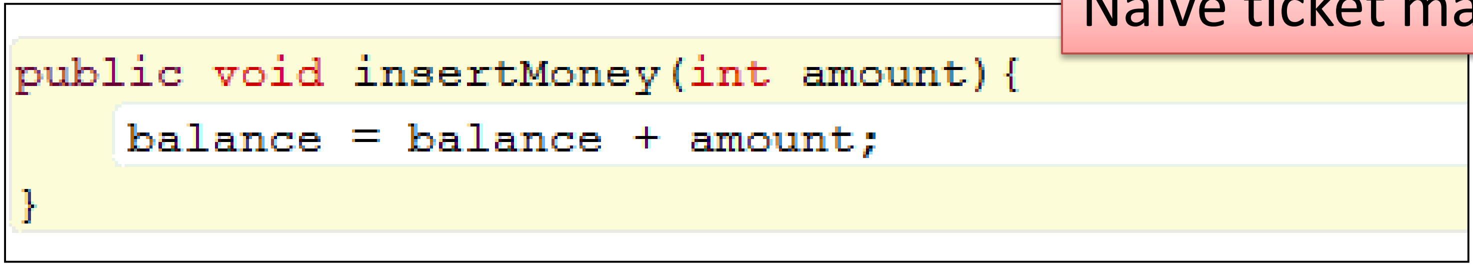
Note: we will look at this in more detail a subsequent lecture.

Topic List

- Data types:
  - primitive
  - objects
- Demo of naïve ticket machine
- Inside classes:
  - fields
  - constructors
  - methods:
    - accessors
    - mutators
  - assignment statements
- Demo of better ticket machine
- Making choices: conditional statements (if)

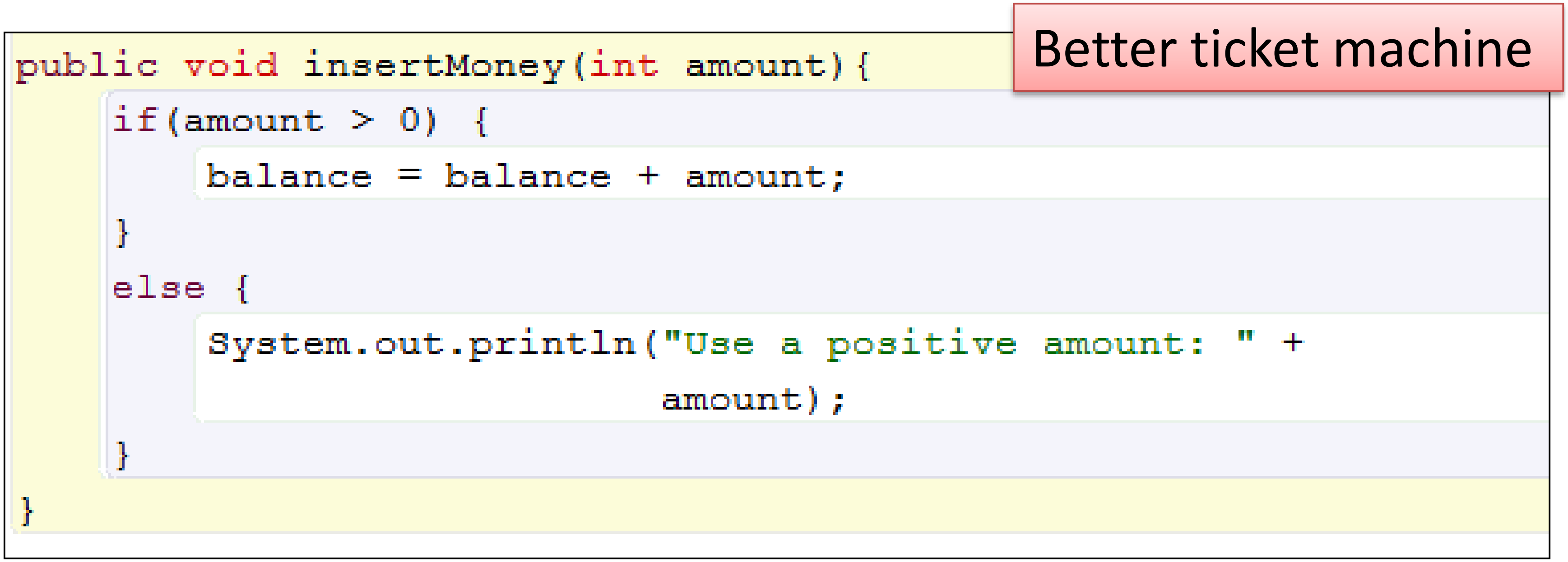
Adding checks by making choices

Naïve ticket machine



```
public void insertMoney(int amount){
    balance = balance + amount;
}
```

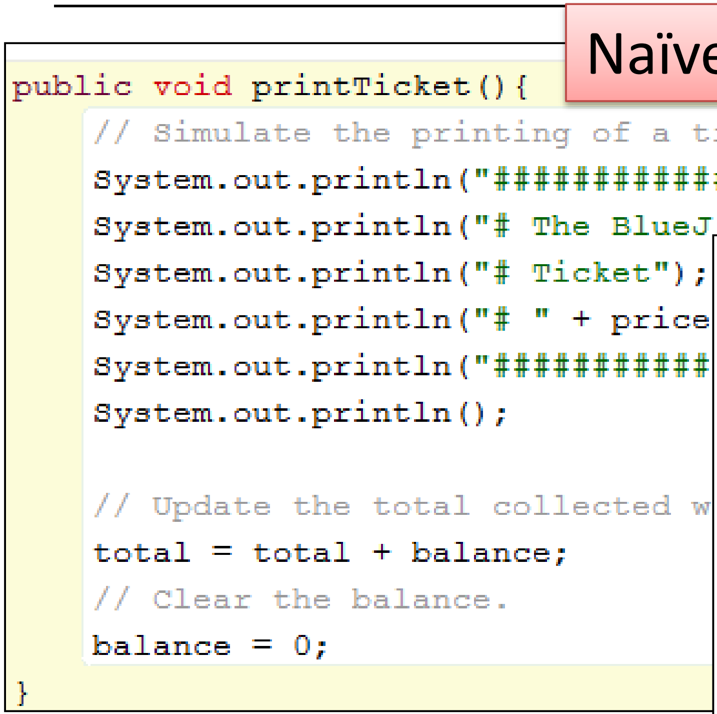
Better ticket machine



```
public void insertMoney(int amount){
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: " +
                           amount);
    }
}
```

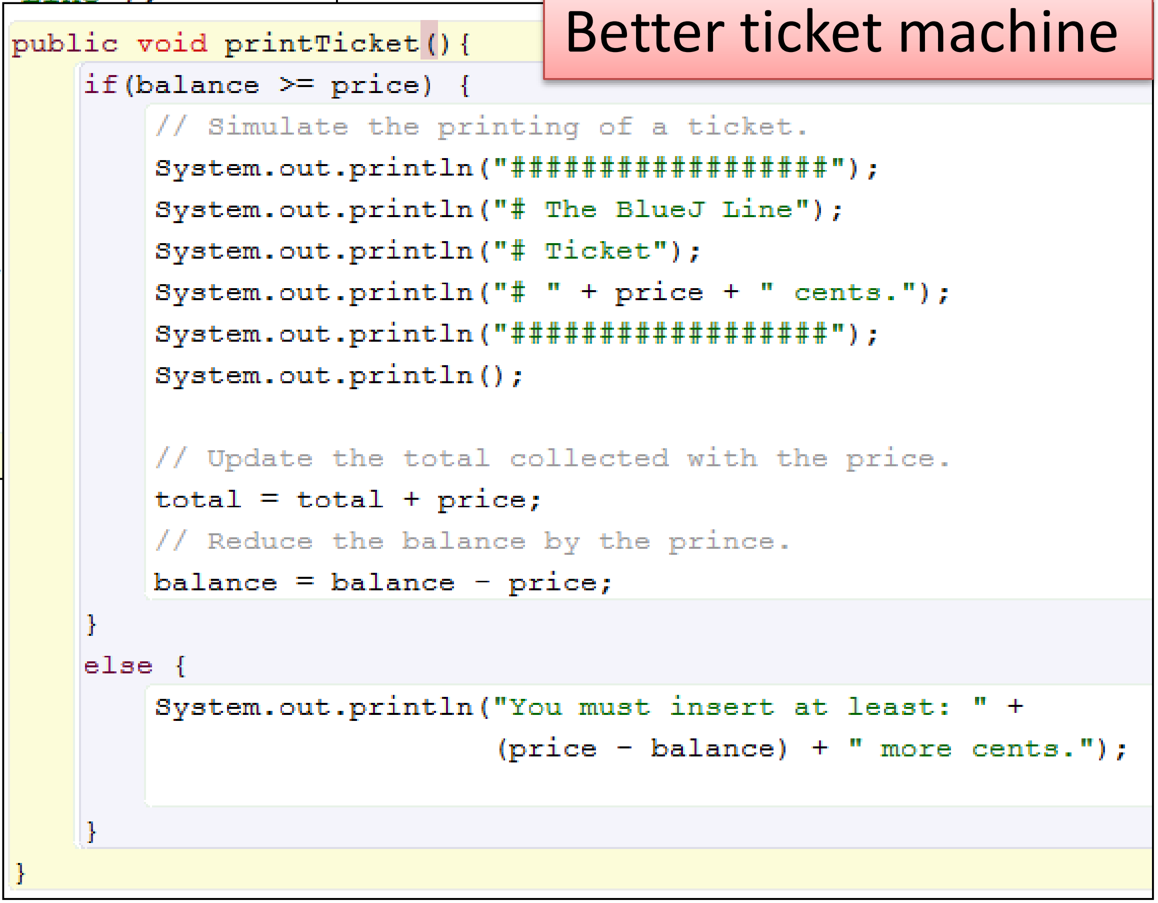
Adding checks by making choices

Naïve ticket machine



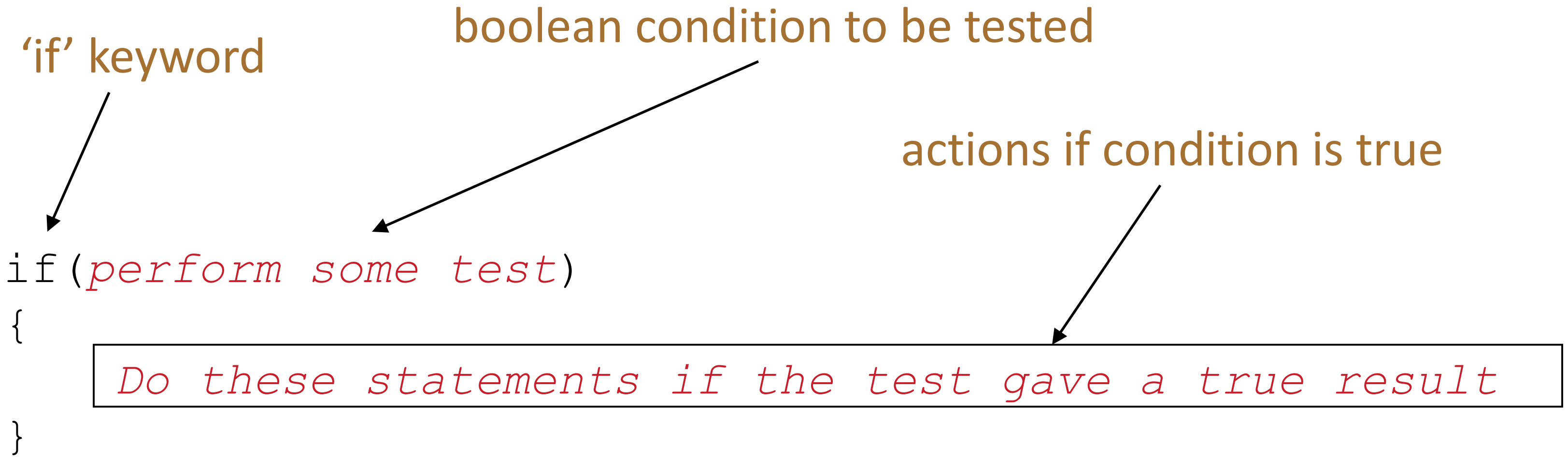
```
public void printTicket(){
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price
    System.out.println("###########
    System.out.println();

    // Update the total collected w
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Better ticket machine



```
public void printTicket(){
    if(balance >= price) {
        // Simulate the printing of a ticket.
        System.out.println("##################");
        System.out.println("# The BlueJ Line");
        System.out.println("# Ticket");
        System.out.println("# " + price + " cents.");
        System.out.println("##################");
        System.out.println();

        // Update the total collected with the price.
        total = total + price;
        // Reduce the balance by the prince.
        balance = balance - price;
    }
    else {
        System.out.println("You must insert at least: " +
                           (price - balance) + " more cents.");

    }
}
```

Conditional Statement Syntax (1)



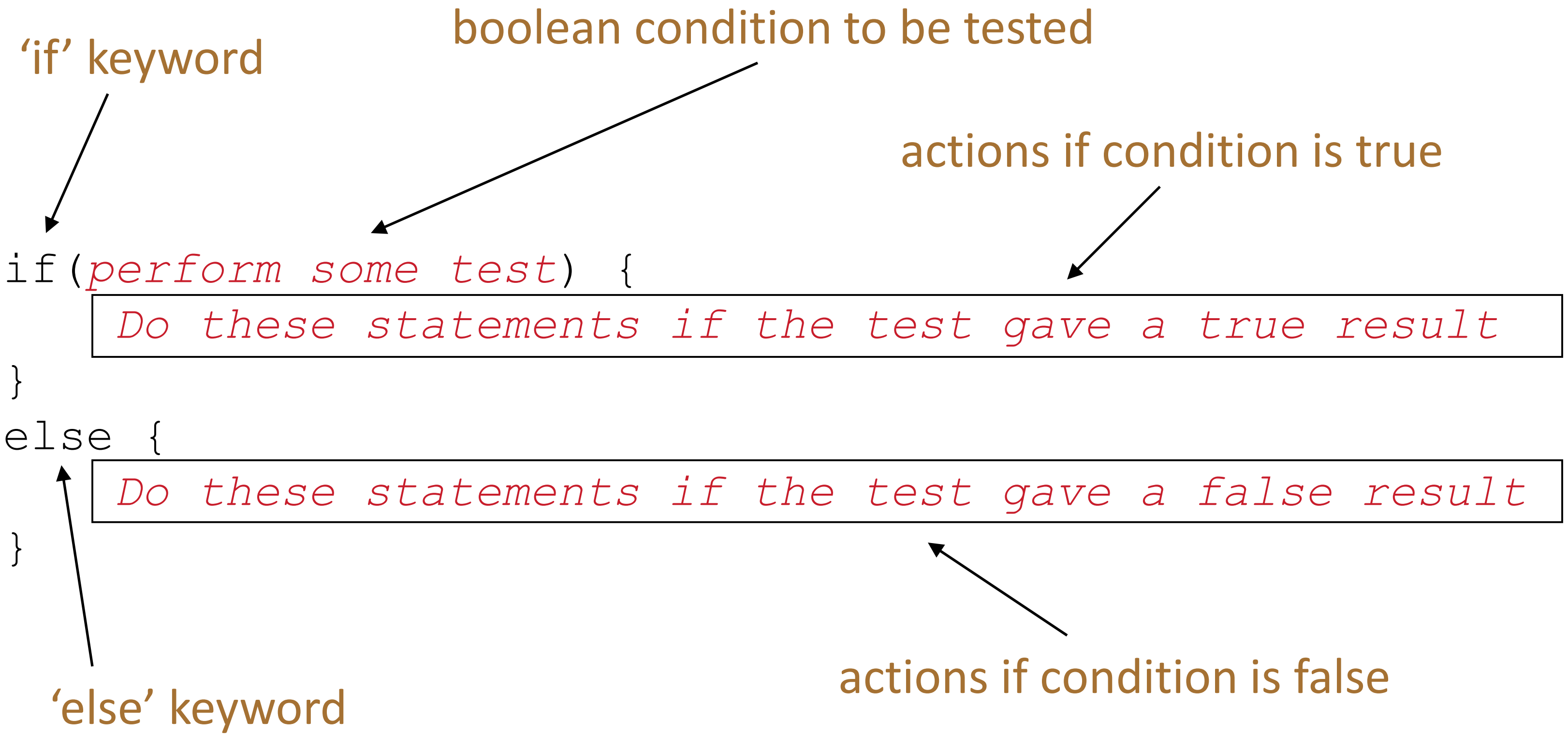
‘if’ keyword

boolean condition to be tested

actions if condition is true

```
if(perform some test)
{
    Do these statements if the test gave a true result
}
```

Conditional Statement Syntax (2)



‘if’ keyword

boolean condition to be tested

actions if condition is true

```
if(perform some test) {
    Do these statements if the test gave a true result
}
else {
    Do these statements if the test gave a false result
}
```

‘else’ keyword

actions if condition is false

Conditional Statement Syntax (3)

```
if(condition1…perform some test)
{
    Do these statements if condition1 gave a true result
}
else if(condition2…perform some test)
{
    Do these statements if condition1 gave a false
    result and condition2 gave a true result
}
else
{
    Do these statements if both condition1 and
    condition2 gave a false result
}
```

Some notes on the if statement

- An if statement IS a statement; it is only executed once.
- When your if statement only has one statement inside it, you do not need to use the curly braces.
- For example, both of these are the same:

```
if (balance >= price)
{
    System.out.print(“Sufficient funds”);
}
```

```
if (balance >= price)
    System.out.print(“Sufficient funds”);
```

Some notes on the if statement

- The semi-colon (;) is a statement terminator.
- One is circled in the code example below:

```
if (balance >= price)
{
    System.out.print(“Sufficient funds”);
}
```

- Your if statement does not need a statement terminator.

Improving the constructor

```
public TicketMachine(int ticketCost)
{
   price = ticketCost;
   balance = 0;
   total = 0;
}
```

```
public TicketMachine(int ticketCost)
{
     if (ticketcost > 0)
     {
         price = ticketCost;
     }
     else
     {
         price = 20;
     }
     balance = 0;
     total = 0;
}
```

Note: in the constructor set the field to a default value if invalid data was entered…maybe our tickets will have a default cost of 20 if an invalid ticketCost is entered.

Improving the setter / mutator

```
public void setBalance(int amount)
{
    if (amount > 0)  {
      balance = amount;
     }
}
```

Note: The validation done at constructor level must be repeated at setter level for that field.

However, in setter methods do not update the field’s value if invalid data was entered (notice how the “else” part of the “if” is not there).

Study aid: can you answer these questions?

- Java has two kinds of types…what are they?
- How many primitive types does Java have? Can you name them?
- Can you give an example of an object type?
- What are instance fields? What does object state mean?
- What is the job of a constructor? How do you recognise one i.e. method signature?
- What is a method signature? What is a method body?

Study aid: can you answer these questions?

- What are accessor methods and how would you recognise them in your source code?
- What are mutator methods and how would you recognise them in your source code?
- What are assignment statements? Can you write a statement that declares a ***String*** variable called ***name*** and updates its contents to ***Joe Soap***?
- What are if statements? How do you write them?
- What are boolean expressions?

Questions?