AP Java chapter 2

Concepts:

Fields	**Methods (accessor, mutator)**
Constructors	**Assignment and conditional statements**
parameters	

Constructs:

field	**constructor**	**parameter**
assignment (=)	**block**	**return statement**
void,	**compound assignment operators (+=,-=)**	**if**

Source code for *TicketMachine (naïve-ticket-machine* on CD)

```
/**
 * TicketMachine models a naive ticket machine that issues
 * flat-fare tickets.
 * The price of a ticket is specified via the constructor.
 * It is a naive machine in the sense that it trusts its users
 * to insert enough money before trying to print a ticket.
 * It also assumes that users enter sensible amounts.
 *
 * @author David J. Barnes and Michael Kolling
 * @version 2003.12.01
 */
public class TicketMachine
{
    // 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;
```

Fields defined

```
    /**
     * Create a machine that issues tickets of the given price.
     * Note that the price must be greater than zero, and there
     * are no checks to ensure this.
     */
    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
        balance = 0;
        total = 0;
    }
```

Constructor initializes variables

```
    /**
     * Return the price of a ticket.
     */
    public int getPrice()
    {
        return price;
    }

    /**
     * Return the amount of money already inserted for the
     * next ticket.
     */
    public int getBalance()
    {
        return balance;
    }

    /**
     * Receive an amount of money in cents from a customer.
     */
    public void insertMoney(int amount)
    {
        balance = balance + amount;
    }
    /**
     * Print a ticket.
     * Update the total collected and
     * reduce the balance to zero.
     */
    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;
    }
}
```

methods

Chapter 2.3 Fields, constructors and methods

- Source code broken into outer wrapping, and inner part.
  - Outer wrapping only names the class.
  - Inner wrapping does all the work. I.e.

```
public class TicketMachine
{
    inner part goes here
}
```

- Inner part of class defines *fields, constructors, methods*
  - Fields = store data for each object to use (also called *instance variables)*
  - Constructors = correctly set up each object when first created
  - Methods = implement behavior of objects.

In TicketMaster:

- Fields = price, balance, total
  - *Price* stores fixed price of a ticket
  - *Balance* stores amounted inserted into machine by user
  - *Total* stores total inserted by all users

```
// 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;
```

Note:
// = comment on single line
/* …*/ comment on multiple lines.

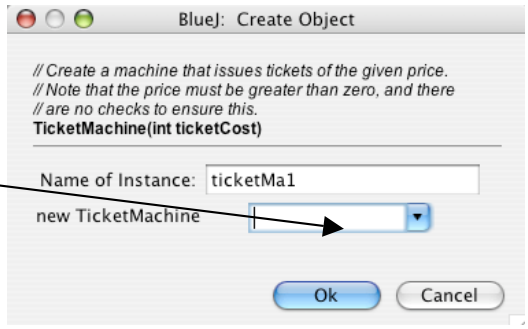
  - Fields are always defined as *private*
  - Field definitions must include type (i.e. int).
  - Fields = space inside an object into which to store values.
- Constructors = (where initialization takes place).

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

  - Constructor initializes object to a reasonable state.
  - Have same name as class in which they are defined.
  - Fields are initialized in constructor. Some set to integer (i.e. 0); some set to parameter to be entered after program is run (i.e. ticketCost).

Chapter 2.4 Passing data via parameters

- Methods and constructors receive values via parameters
- Parameters defined in header of constructor
  - i.e. public TicketMachine(int ticketCost)
  - Value entered into parameter *ticketCost* when object is created:
  - *tickteCost* labels *constructor space* = area inside constructor for values.
  - Value entered into field, *price*,
  - *Formal Parameters* = names of parameters outside (ticketCost)
  - *Actual Parameters* = values placed inside parameters. (500)
  - *Scope* = section of source code from where variable can be accessed.
  - *Lifetime* = how long variable continues to exist before it is destroyed. Life of parameter is limited to single call of constructor or method. After constructor call, parameter loses values. Fields (*price*) retain values.



Chapter 2.5 Assignments:

- Assignment Statements store value on right side of statement in the variable named on the left. (**price = ticketCost**) ticketCost goes into price . price <= ticketCost.
- Generic description: Variable = expression
- Expression =
  - compute values to place into variable
  - Expression type must match variable type

Chapter 2.6 Accesssor Methods:

- Methods have two parts:
  - Header

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

    - Blue lines = comment describing what method does
    - Last line = *method signature*
      - *Different from field declaration because of: following parenthese, no sem colon.*
  - Body
    - Remainder of method after header
    - Enclosed in { } declarations and statements inside {} called a *block*
    - Contain declarations and statements what happens when method is called.

- *Differences between signatures of constuctors (**public TicketMachine (int TicketCost)** and methods (**public int getPrice()***
    - *Method has return type of int, constructors (TicketMachine) can not have a return type. (parameter, TicketCost has type of int)*
    - *Constructors and methods can have 0 to any amount of parameters*

- Statement in body = **return price;**
    - Called *return statement*
    - Return statement returns value of type stated in method signature.
    - Return statement is always last statement in a method.

- Method call = question to the object. (what is price of ticket?); return statement provides answer. (shows contents of field *price*.
- Accessor Methods: return information about the state of an object.
    - Provide access to that state.
    - Usually in form of return statement (also by printing information).

Chapter 2.7 Mutator Methods

- Mutator Methods change the value of one or more fields. Change the state of the object.
- Objects exhibit different behavior before and after mutator is called.
- *Void* return type means that method does not return any value to caller.

Chapter 2.8 Printing Methods

- The method **System.out.println** is a built in method that prints to terminal.
- Format = System.out.println (" text to be printed");
- Format to concatenate strings and values = System.out.println ("text " + field + " text.");

Chapter 2.11 Conditional statements (better-ticket-machine – on CD)

- Conditional statement;
    - Takes action based on result of a test.
    - Also called IF STATEMENTS.
    - Type of BOOLEAN EXPRESSION: only returns TRUE or FALSE result.

    - Form =

```
If (perform tests) {
      Perform action if test is true
}
else{
      Perform action if  test is false
}
```

Sample from better-ticket-machine:

```
/**
*Receive an amount of money in cents from a customer.
* Check that the amount is sensible.
*/
public void insertMoney(int amount)
   {
      if(amount > 0) {
         balance = balance + amount;
      }
      else {
         System.out.println ("Use a positive amount: " +
                        amount);
      }
```

Chapter 2.12 another conditional statement

- Shows how to add print and math operation to conditional statement

```
/**
    * Print a ticket if enough money has been inserted, and
    * reduce the current balance by the ticket price. Print
    * an error message if more money is required.
    */
   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.");

      }
   }
```

Chapter 2.13 Local variables

- Local variable:
  - Variable declared and used in a single method. Scope and lifetime are limited to method.

```
public int refundBalance()
  {
    int amountToRefund;
    amountToRefund = balance;
    balance = 0;
    return amountToRefund;
  }
```

    - AmountTo Refund = local variable
    - declared in – *int amountToRefund*;
    - initialized – *amountToRefund = balance;*
    - Can combine – *int amountToRefund = balance;*
  - Never use private or public in declaration.

Chapter 2.14 Fields, Parameters, local variables

- All three types of variables: Store values appropriate to defined type
- **Fields:**
  - Defined outside constructors and methods
  - Store data that lives through life of object, have lifetime as long as object's lifetime, maintain current state of object.
  - Have class scope – extends through class. Can be used in any method or constructor in class.
  - Private fields cannot be accessed outside of class.
- **Formal Parameters:**
  - Exist only for the time a constructor or method executes.
  - Values lost between calls.
  - Temporary storage areas
  - Defined in header of constructor or method
  - Receive values from outside.
  - Scope limited to defining constructor or method

- **Local variables:**
  - Defined inside body of constructor or method
  - Must be initialized only inside constructor or method . Receive no default initialization.
  - Exist only for the time a constructor or method executes.
  - Values lost between calls.
  - Temporary storage areas
  - Scope limited to block in which they are defined.

Chapter 2.16 (use lab-classes from CD)

- Substrings allow to return parts of string (i.e. first 4 characters).
- From lab-classes:

```
/**
     * Return the login name of this student. The login name is a combination
     * of the first four characters of the student's name and the first three
     * characters of the student's ID number.
     */
    public String getLoginName()
    {
        return name.substring(0,4) + id.substring(0,3);
    }

    /**
     * Print the student's name and ID number to the output terminal.
     */
    public void print()
    {
        System.out.println(name + " (" + id + ")");
    }
}
```

- name.substring (0,4) places first 4 characters of string into field, name.
- Returns: characters from beginIndex to (endindex – 1). First character is 0.

Summary:
Vocab: field, comment, constructor, scoope, lifetime, assignment, method, accessor method, mutator method, println, conditional, Boolean expression, local variable.