Introduction to Java

Handout-1c

Java Applications

- Type the following program into a file named HelloWorldCommandLine.java and save the file in the directory where you do your work (C:\cs402\)

```
public class HelloWorldCommandLine {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Program execution

- Compile:
  - C:\cs402> javac HelloWorldCommandLine.java
  - The output is the Java bytecodes file named HelloWorldCommandLine.class
- To execute call the Java interpreter:
  - C:\cs402> java HelloWorldCommandLine

Syntax rules

- Comments

```
// This is a comment

/* this is the first line of a multi-line comment
More comments here.
*/
```

- White-space: Empty lines, space, tabs do not affect the program

Syntax rules (ii)

- The program starts with *import statements* followed by a *class definition*
  - These are user defined classes
- The work `class` is a *keyword*
  - Always lower case
  - Keywords are reserved words by java
- Another keyword is `public`
- Ex: `public MyClass { }`
  - it indicates that the class can be accessed from other classes

Syntax rules (iii)

- The name of a class is an *identifier*
  - An identifier may not begin with a digit and may not have any spaces in it
  - Ex: `public MyClass { }`
  - The name of the file must be MyClass.java
- Keywords may not be used as identifiers
- Java is case sensitive
- By convention, the name of a class must begin with a capital letter

Syntax rules (iv)

- Import statements
  - All import statements are at the beginning of the file, before any class declaration
- Ex: `import java.awt.*;`
  - This will import all (this is the meaning of *) classes in the package awt
  - You can import only a specific class:
  - Ex: `import java.awt.event;`
- The package `java.lang` is automatically imported in every program (you don't have to use an import statement)

Syntax rules (v)

- Inside the class brackets we define *class members*:
  - Data fields
  - Methods (functions)
- Ex:

```
public class MyClass {
    int someIntegerData;

    int someMethod() {
    …
    }
}
```

- The example defines a class with two members

What is OOP?

- Object-oriented programming is 30+ years old
  - Was introduced with Simula-67
- Four key principles
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

Abstraction (i)

- Needed if we are to represent real-world objects in a computer
  - We need to extract the essential characteristics of an entity. The data representing these characteristics is what will be used for processing in a computer
- Ex: car
  - Registration authority
  - Home records
  - Garage records

Abstraction (ii)

- Data abstraction is the process of refining away the unimportant details of an object, such that only the appropriate characteristics that describe it remain
- Essential object characteristics together with the operations on the data form an *abstract data type*

Encapsulation

- Encapsulation associates data and the operations that can be performed on data in a single unit of organization (*class*)
- Non-OOP languages (C, Perl, etc.) support encapsulation for built-in types, but not for user-defined types
- OOP languages support encapsulation on built-in data types AND user defined types
  - Operations on user-defined types tend to be expressed as functions, aka *methods*

Java built-in (primitive) data types

- There are eight built-in types in Java
  - `byte` – one byte, 2's complement representation
  - `short` – two bytes, 2's complement representation
  - `int` – four bytes, 2's complement representation
  - `long` – eight bytes, 2's complement representation
  - `float` – four bytes, IEEE-754 standard
  - `double` – eight bytes, IEEE-754 standard
  - `char` – two bytes, unsigned integer, Unicode
  - `boolean` – true or false values

Classes (i)

- *Class* is another word for ‘user-defined type’
- Defining a class

```
class ClassName { classMembers }
```

- Defining a class doesn’t bring any of its objects into existence. Remember, a class is just the definition of type
- Class members:
  - Data
  - Functions (methods)
- Objects of a class are known as *instances*

Classes (ii)

- Example of a class:

```
class Fruit {
  int grams;

  int totalCalories() {
    return grams*10;
  }
}
```

- The class has two members
  - A data field named `grams` of type integer
  - A method called `totalCalories` that takes no arguments and returns an integer

Classes (iii)

- Variables of a class are called *objects*
- Ex:

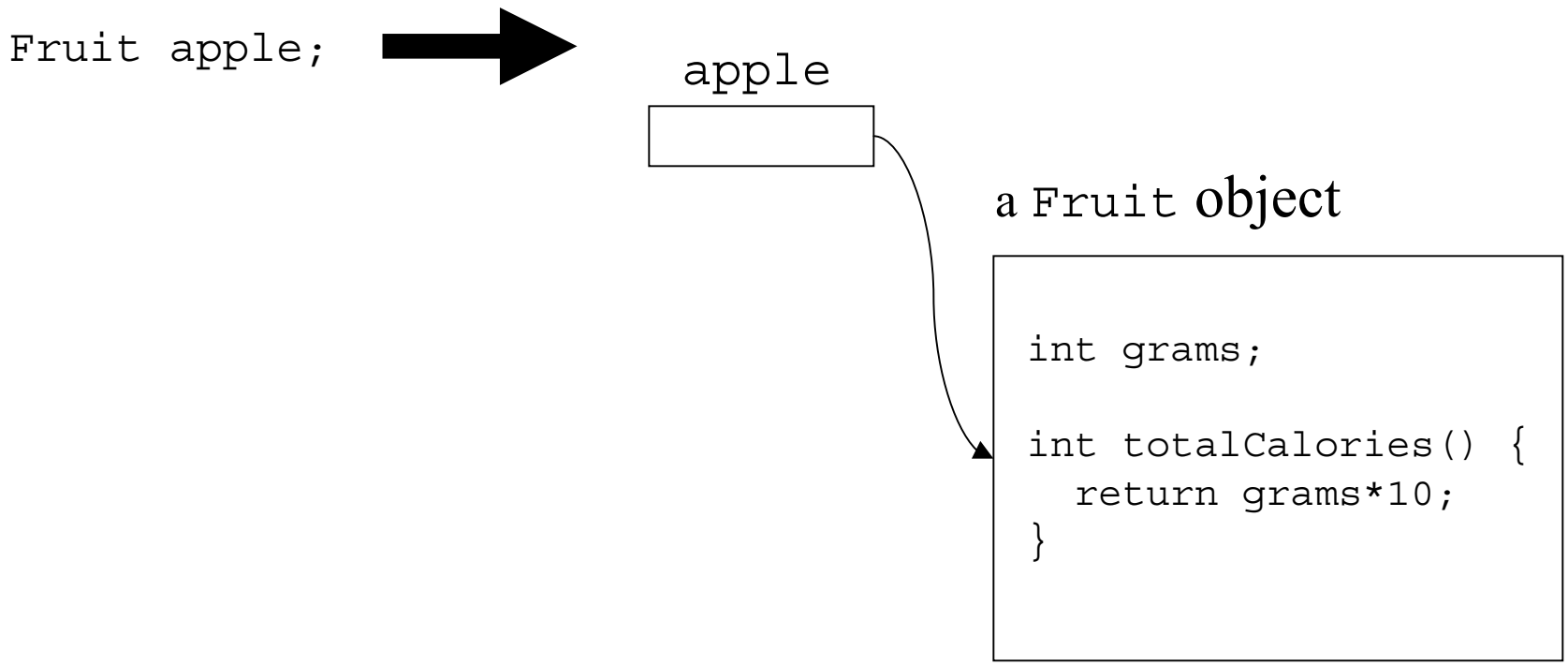
```
Fruit apple, orange, a;
```

- This declares three objects, `apple`, `orange`, and `a` to be of type `Fruit`
- The objects are NOT created when they are declared:
  - `apple` is a variable that is a *reference* to an object of type `Fruit`.
  - When declared, the reference is a null pointer

Classes (iv)

The declaration of an object variables creates a place in memory that *can* hold a reference to an object

means reference to an object



Classes (v)

- Semantic difference between primitive data types and variables of a class type:
  - When declaring a variable of primitive type, the compiler allocates memory and you can start using it right away
  - When declaring a variable of class type, the compiler reserves space in memory for a reference to an object of that class and initializes the reference to null

Classes (vi)

- Operations on Objects: use the "dot" notation to access the members of a class:
  - Ex: `apple.grams`
  - Ex: `apple.totalCalories()`
- Multi-level references are possible
  - Ex: `a.b.c.d()`

Classes (vii)

- Creating new objects: *constructors*
- A constructor is a special kind of method that you write as part of a class
  - Creates object
  - Initializes object
- Has the same name as the class
  - Ex: `Fruit myBasket = new Fruit();`
- The variable `myBasket`, of type `Fruit`, now holds a reference to the object created by calling the constructor `Fruit()`

Classes (viii)

- Here is the Fruit class with some constructors

```
Class Fruit {
  int grams, caloriesPerGram;
  int totalCalories() {
    return grams*10;
  }

  Fruit() {       // constructor
    grams = 10;
    caloriesPerGram = 0;
  }

  Fruit(int g, int c) {       // another constructor
    grams = g;
    caloriesPerGram = c
  }
}
```

Classes (ix)

- If you don't provide an explicit constructor, then the *default no-arg* constructor will be used:
  - Takes no arguments
  - Does nothing
  - Ensures that each class has at least a constructor

Classes (x)

- When the constructor is called:
  - Memory for the object is allocated
  - Memory allocated for object is initialized with default values (zero, null, 0.0, etc.). All objects start with a known state

OOP key principles

- Abstraction √
- Encapsulation √
- Inheritance
- Polymorphism

Inheritance (introduction)

- A class can be related to another class in a parent-child relationship
- The child *extends* the parent class with additional members or changes
- Ex:

```
class Fruit {}
class Citrus extends Fruit {}
…
Fruit lime = new Citrus();
```

- Citrus is a *child class* of Fruit, aka *subclass* or *subtype*

Inheritance (ii)

- Inheritance in OOP is “what you get from the parent class”
- All classes have a parent class
  - All objects in the system are subtypes of `java.lang.Object` and have all the members of that class

```
class A { … }
```

really means

```
class A extends java.lang.Object { … }
```

Inheritance (iii)

- **Class**: a data type
- **Extend**: to make a new class that inherits the contents of an existing class
- **Superclass**: a parent or "base" class. The word wrongly suggests that the parent class has more than the subclass. It means "super" in the sense of "above"
- **Subclass**: a child class that inherits, or extends a superclass. It is called subclass because it only represents a subset of the universe of things represented by the superclass

Inheritance (iv)

- A constructor in the object's parent class is *always* called
  - This is done recursively (all the way back to the Object class)
  - Constructing new object can be expensive