

Introduction to Programming

with Java, for Beginners

GUI
Casting
++ & -- operator
Switch statement
Main with arguments
Heap Management

Intro to Graphical User Interface(GUI)

- Create a window in which to display things—usually a `JFrame` (for an application), or a `JApplet` (for web-browser)
- Use the `setLayout(LayoutManager manager)` method to specify a layout manager
- Create some `Components`, such as buttons, panels, etc.
- Add your components to your display area, according to your chosen layout manager
- Write some Listeners and attach them to your `Components`
 - Interacting with a Component causes an `Event` to occur
 - A Listener gets a message when an interesting event occurs, and executes some code to deal with it

ESE112

2/15

Necessary Packages to Import

- The GUI components are in `javax.swing.*`, so you always need to import that for a Swing application
- Swing is built on top of AWT (Abstract Window Toolkit) and uses a number of AWT packages, including most of the layout managers, so you need to import `java.awt.*`
- Most listeners also come from the AWT, so you also need to import `java.awt.event.*`

ESE112

3/15

Events and Listeners

- Interacting with a GUI component (such as a button) causes an event to occur
- An `Event` is an object in Java
- You create `Listeners` for interesting events
 - `Listener` is an *interface*; you create a `Listener` by implementing that interface
- The `Listener` method gets the `Event` as a parameter

ESE112

4/15

actionPerformed Method

```
public void actionPerformed(ActionEvent event){..}
```

- If there are multiple GUI components then we want to know which button was pressed and what to do with them
- `event.getSource()` return reference (heap address) to the object (component) that caused the event
- ```
if(event.getSource() == button1) {
 //do something
}
else if (event.getSource() == button2){
 //do something different
}
```

ESE112

5/15

## Casting with Numeric Primitive Types

- Numeric types are considered wider or narrower than other numeric types
  - Based how much memory space they occupy
- Java doesn't mind if you assign a narrow value to a wide variable: `double n = 3;`
- Java is not happy if you assign a wide value to a narrow variable: `int n = 3.5; //illegal`
- But if you want to narrow (assign a wider type to a narrower type), you have to **cast** it:  
`double d = 3.5;  
i = (int) d; //legal due to casting`
- Java checks to make sure that the cast works, and gives you an error if it didn't

ESE112

6

## Casts with Reference Types

```
public class Vehicle{
 protected int regNum;

 public Vehicle(int r){
 regNum = r;
 }

 public int getRegNum(){
 return regNum;
 }
}

public class Car extends Vehicle{
 protected int numDoors;

 public Car(int r, int n){
 super(r); //or regNum = r
 numDoors = n;
 }

 public int getDoors(){
 return numDoors;
 }
}
```

ESE112

7

## Casts with Reference Types (contd..)

```
> Vehicle v = new Car(45,4); //valid
> v.getRegNum()
45
> v.getDoors()
Error
```

- Illegal because "v" could potentially refer to other types of vehicles that are not cars
- The solution here is to use type-casting.

ESE112

8/15

## Casts with Reference Types (contd..)

- If, for some reason, you happen to know that “v” does in fact refer to a Car, you can use the *type cast*
  - Use **instanceof** keyword to find that out
- Do (Car)v to tell the computer to treat “v” as if it were actually of type Car. So, you could do:  
(Car)v.getDoors()

ESE112

9

## The *increment* operator

- **++** adds 1 to a variable
  - It can be used as a statement by itself, or within an expression
  - It can be put *before* or *after* a variable
  - If before a variable (pre-increment), it means to add one to the variable, then use the result
  - If put after a variable (post-increment), it means to use the current value of the variable, then add one to the variable
- The same applied to decrement operator

ESE112

10

## Examples of ++

```
int a = 5;
a++;
// a is now 6

int b = 5;
++b;
// b is now 6

int c = 5;
int d = ++c;
// c is 6, d is 6

int e = 5;
int f = e++;
// e is 6, f is 5

int x = 10;
int y = 100;
int z = ++x + y++;
// x is 11, y is 101, z is 111
```

Confusing code is bad code, so this is very poor

style

ESE112

11

## char

- The primitive type **char**
  - Just stored as numbers
  - Each char as a unique integer value (based on Unicode standard)
- You can use characters in arithmetic (they will automatically be converted to **int**)
  - > char ch = 'A';
  - > ch + 1
  - 66
  - > char ch2 = (char) (ch + 1) // cast result back to char
  - B

ESE112

12

## Syntax of the *switch* statement

- The syntax is:

```
switch (expression) {
 case value1 :
 statements ;
 break ;
 case value2 :
 statements ;
 break ;
 ...(more cases)...
 default :
 statements ;
 break ;
}
```
- The *expression* must yield an integer or a character
- Each *value* must be a literal integer or character
- Notice that colons ( : ) are used as well as semicolons
- The last statement in every case should be a *break*;
  - I even like to do this in the *last* case
- The *default*: case handles every value not otherwise handled

ESE112

13

## Example switch statement

```
switch (cardValue) {
 case 1:
 System.out.print("Ace");
 break;
 case 11:
 System.out.print("Jack");
 break;
 case 12:
 System.out.print("Queen");
 break;
 case 13:
 System.out.print("King");
 break;
 default:
 System.out.print(cardValue);
 break;
}
```

ESE112

14

## Main

```
public static void main (String [] args)
```

- Must have the exact signature
  - Only variation allowed is name of the input parameter
- So main starts everything, how do we call main and provide inputs ?
- To run a program recall
  - Command: *java ClassName*
    - This what calls the main method if the class has one
  - So we could pass arguments as follows:  
*java ClassName list-of-arguments*

ESE112

15

## Main with arguments example

```
public class ExampleArgs{
 public static void main(String [] args){
 System.out.println("Demo for Inputs args");
 for(int i = 0; i < args.length; i++){
 System.out.println(args[i]);
 }
 }
}
```

```
> java ExampleArgs ESE 112
Demo for Inputs args
ESE
112
```

Note: Code works even if no arguments are passed to main() because JVM passes to main() a zero-length array of Strings and not a null

ESE112

16

## Memory Management

- Memory is not infinite
- Stacks grow and shrink
- Heap
  - Grow when you dynamically allocate memory i.e. new Object()
  - If you do not manage the allocations then you will run out of this memory
    - Objects that will never be accessed or mutated again by application need to be reclaimed
      - This is known as *Garbage Collection*
- Some Languages like C/C++ leave it up to the programmer to do explicit memory management
- Java does automatic garbage collection
  - Done by JVM (Java Virtual Machine)