

Rad sa vremenom

- U Javi, rad sa vremenom možemo podeliti na pre i posle pojavljivanja Jave 8
- U prethodnim verzijama Jave, sistem za rad sa vremenom bio je zbunjujući i ne baš naročito dobro realizovan.
- Nakon Jave 8, ovaj sistem je obogaćen novim i boljim sistemom
- Koji god sistem koristili, rad sa vremenom će podrazumevati neku od sledećih akcija:
 - preuzimanje tačnog vremena
 - Dodavanje / oduzimanje vremena
 - Pretvaranje vremena u string radi prikaza
 - Pretravaranje vremena iz stringa, radi obrade

Rad sa vremenom do jave 8

- Ukoliko će naš projekat postojati na sistemu koji ima verziju Jave manju od 8 (ili jednostavno nismo sigurni), tada treba koristiti sledeće klase za rad sa vremenom:
 - `java.util.Date`
 - `java.util.Calendar`
 - `java.util.GregorianCalendar`

Rukovanje milisekundama

- Jedino što nam je zapravo potrebno da bi radili sa vremenom jeste trenutni broj milisekundi koje su protekle od 01.01.1970. godine
- U Javi imamo mogućnost da dobavimo ovu informaciju na više načina
- U Javi, broj milisekundi se može dobiti pomoću:
 - **`System.currentTimeMillis()`**
 - Naredba daje rezultat poput sledećeg: *1415559852660*
- Sledeći kod prikazuje broj milisekundi u mrtvoj petlji

```
while(true) {  
    System.out.print(System.currentTimeMillis()+"\r");  
}
```

- Milisekunde takođe možemo dobiti metodom **`getTime`**, klase **`Date`**.

```
Date dt = new Date();  
System.out.println(dt.getTime());
```

Klasa Date

- Klasa Date je osnovna klasa za rad sa vremenom
- Ona je u stanju da sačuva datum i vreme
- Nakon jednostavnog kreiranja, ova klasa u sebi nosi datum i vreme trenutka u kome je kreirana
- Većina metoda ove klase su označene kao **deprecated**, ali se ipak još uvek mogu koristiti
- Objekat klase Date se kreira na standardan način:

```
Date date = new Date();  
System.out.println(date);
```

- Ovako kreiran objekat sadržaće vreme za trenutak u kome je kreiran, a njegova string reprezentacija izgledaće ovako:

```
Mon Nov 10 08:34:57 CET 2014
```

- Najčešće nećemo hteti ovakav izgled na izlazu

Klasa Date – izlazno formatiranje

- Da bi vreme prikazali na izlazu onako kako nama odgovara, koristimo klasu **SimpleDateFormat** (metod **parse**).
- Recimo da kreirani datum hocemo da formatiramo na sledeći način: MESEC/DAN/GODINA SATI:MINUTI:SEKUNDE, mogli bi napisati:

```
Date date = new Date();  
SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/y HH:m:s");  
System.out.println(sdf.format(date));
```

- Vidimo da se za formatiranje koriste odgovarajući karakteri. Ove karaktere ne moramo znati napamet (iako se vrlo brzo nauče)
- <https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html>

Klasa Date – ulazno formatiranje

- Osim formatiranja vremena na izlazu, često ćemo imati problem sa vremenom na ulazu. Iz različitih formata zapisa datuma, trebaće nam validan Date objekat. Ovde takođe možemo koristiti SimpleDateFormat
- Recimo da imamo sledeći string: „2014 december 15“ i da iz njega hoćemo da izvučemo i kreiramo datum. Mogli bi napisati:

```
SimpleDateFormat sdf_input = new SimpleDateFormat("yyyy MMM d");  
String some_string_date = "2014 december 15";  
Date date_input = sdf_input.parse(some_string_date);  
System.out.println(date_input);
```

- Na izlazu će rezultat biti sledeći:

```
Mon Dec 15 00:00:00 CET 2014
```

- *Primećujemo da je Date postavio podrazumevane vrednosti za delove vremena za koje nije imao dovoljno informacija*

Klasa Calendar

- Za “naprednije” rukovanje vremenom, koristi se klasa **java.util.Calendar**
- Klasa Calendar je apstraktna, pa se obično koristi posredstvom klase **GregorianCalendar**

```
GregorianCalendar cal = new GregorianCalendar();  
System.out.println(cal.getTime());
```

Klasa Calendar – postavljanje vremena

- Vrednosti se objektu klase Calendar postavljaju metodom **set**, a preuzimaju metodom **get**.
- Kod metode **set**, prvi parametar je uvek konstanta (celobrojna vrednost) koja predstavlja polje na koju se vrednost odnosi, dok je drugi parametar vrednost
- Kod metode **get**, samo se prosleđuje konstanta sa nazivom željenog polja

```
GregorianCalendar cal = new GregorianCalendar();  
cal.set(Calendar.YEAR, 2014);  
cal.set(Calendar.DATE, 15);  
cal.set(Calendar.MONTH, 11);  
System.out.println(cal.getTime());  
System.out.println("The exact month is " + (cal.get(Calendar.MONTH)+1));
```


Klasa Calendar – dodavanje i oduzimanje vremena

- Klasa Calendar ima mogućnost dodavanja vremena na postojeće vreme, ili oduzimanja vremena od postojećeg
- Dodavanje ili oduzimanje vremena vrši se metodom **add**, po sličnom principu kao i postavljanje vremena
- Recimo da hoćemo da proverimo da li je od nekog perioda u vremenu prošlo 24 časa

```
GregorianCalendar source_date = new GregorianCalendar(2014,10,8);
GregorianCalendar current_date = new GregorianCalendar();
current_date.add(Calendar.HOUR, -24);
System.out.println(current_date.before(source_date));
```

Vežba 3 (jcex112014 FpsCounter)

- Potrebno je kreirati program koji će izvršavati određeni metod u određenom vremenskom intervalu (na primer 30 puta u sekundi)
- Metod koji treba pozivati se može zvati **tick**
- Unutar metode tick, treba prikazati trenutnu brzinu izvršavanja (broj frejmova po sekundi)

Java 8 rad sa vremenom

- Java 8 donosi inovacije upravo u radu sa vremenom, u vidu kompletnog novog API-ja
- Ako projekat dozvoljava upotrebu Jave 8, onda svakako treba koristiti novi API, u suprotnom, kao alternativno rešenje može se koristiti biblioteka **joda time** (<http://www.joda.org/joda-time/>). Ova biblioteka (koja nije u sklopu Jave) zapravo je uzor za kreiranje novog sistema za rad sa vremenom u Javi 8.
- Klase za rad sa vremenom nalaze se u paketu: **java.time** (<https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html>)

Klasa LocalDate

- Klasa LocalDate ne može se instancirati direktno, već samo putem statičke metode
- Sledeća linija kreira objekat klase LocalDate (postavljenu na vreme kreiranja), metodom now():

```
LocalDate now = LocalDate.now();
```

- Po kreiranju objekta, pomoću različitih metoda dobavljamo odgovarajuće vrednosti:

```
LocalDate now = LocalDate.now();  
System.out.println("Current date: " + now);  
System.out.println("Current month: " + now.getMonth() + " (" + now.getMonthValue() + ")");  
System.out.println("Current year: " + now.getYear());  
System.out.println("Current day: " + now.getDayOfMonth() + " (" + now.getDayOfWeek() + ")");
```

Korisničko definisanje vrednosti

- Ako hoćemo da instanciramo klase `LocalDate` ili `LocalTime` na osnovu korisnički definisanog vremena, koristimo metod **of**.

```
LocalTime userDefinedTime = LocalTime.of(14,10,25);  
System.out.println(userDefinedTime);
```

- Neki od elemenata ovog API-ja, preuzeti su od starog, kao na primer, metod `format`, koji se upotrebljava na isti način kao i istoimeni metod `format` starije klase `SimpleDateFormat`:

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("H-m-s");  
System.out.println(formatter.format(timeNow));
```

Vežba

- U aplikaciju ulaze podaci sa gps uređaja
- Za svaku tačku, gps uređaj šalje sledeću poruku:
- 22052014,44.756364,20.412598,051230,123143124122
- Podaci u poruci su (respektivno): datum, latituda, longituda, vreme i imei uređaja
- Potrebno je kreirati klasu koja je u stanju da parsira podatke i čuva informacije o svim primljenim podacima u smislenim tipovima (na primer, datum da bude tipa `LocalDate`)
- Iako većina uređaja šalje poruke kao što je ova u primeru, neki uređaji šalju poruke u kojima datum i vreme imaju izmenjene pozicije. Ovakvi uređaji šalju još jedan dodatni podatak (oznaku H) na kraju poruke:
- (051230,44.756364,20.412598,22052014,123143124122,H)

Rukovanje periodom

- Moguće je preuzeti period između dva datuma ili dva vremena
- U ovu svrhu koriste se klase **Duration** i **Period**

```
//Getting duration
Duration d = Duration.between(userDefinedTime, timeNow);
System.out.println(d);

//Getting period
Period p = Period.between(LocalDate.of(2014, Month.MARCH, 20), now);
System.out.println(p);
```

Klasa Instant

- Ova klasa omogućava rukovanje vremenom na najnižem nivou dostupnom programu, a to su nanosekunde
- Instant objekat možemo instancirati izjavom: **Instant.now()**
- Takođe, objekat instant se može instancirati pomoću postojećeg timestamp-a (bez obzira na koji način smo do njega došli i koja mu je vrednost):

```
System.out.println(Instant.ofEpochMilli(new Date().getTime()));
```

- Instancirana klasa Instant nudi mnoštvo pomagala za rad pre svega sa celobrojnom reprezentacijom vremena (timestampom)

Upotreba klase Instant

- Jedna od funkcionalnosti klase Instante je I dobavljanje razlike između dva termina (metod **until**)
- Kod sa desne strani predstavlja rešenje malopredložjenog primera sa frame-ovima, ali ovaj put uz pomoć klase Instant

```
Instant start = Instant.now();
Instant end, seconddiff = Instant.now();
float etafps = 1000f/30f;
int current_frame = 0, fps = 0;
while(true) {
    end = Instant.now();
    if(start.until(end, ChronoUnit.MILLIS) >= etafps) {
        start = Instant.now();
        fps++;
    }
    if(seconddiff.until(end, ChronoUnit.MILLIS) >= 1000) {
        System.out.print(fps + "\r");
        fps = 0;
        seconddiff = Instant.now();
    }
}
```

Klasa LocalTime

- LocalTime funkcioniše isto kao i LocalDate, ali se odnosi na vreme, umesto na datum

```
LocalTime timeNow = LocalTime.now();  
System.out.println("Current hour: " + timeNow.getHour());  
System.out.println("Current minute: " + timeNow.getMinute());  
System.out.println("Current second: " + timeNow.getSecond());  
System.out.println("Current nanosecond: " + timeNow.getNano());  
System.out.println(timeNow);
```

Enumeracije


- Enumeracija predstavlja posebnu strukturu čiji je zadatak isključivo skladištenje konstanti nekog tematskog konteksta
- Enumeracije su opšte prihvaćen koncept bez koga se može ali je rad sa njima daleko prijatniji
- Koncept enumeracija malo je kasnije stigao u Javu, ali je sa druge strane sama implementacija ovog sistema bogatija od implementacija istog sistema na drugim platformama

Zašto enumeracija?

- Na primer, ako bi imali klasu Game, hteli bi na neki način da predstavimo stanja te igre (recimo da postoji tri stanja: running, paused i stopped), stanja te igre bi mogla biti predstavljena na sledeći način, pomoću konstanti:

```
public class Game {  
    public final static int RUNNING = 1;  
    public final static int PAUSED = 2;  
    public final static int STOPPED = 3;  
}
```

- Zatim bi mogli koristiti kreirane konstante na sledeći način:



```
int GameStatus = Game.RUNNING;  
switch (GameStatus) {  
    case Game.STOPPED:  
        System.out.println("Game is stopped");  
        break;  
    case Game.PAUSED:  
        System.out.println("Game is paused");  
        break;  
    case Game.RUNNING:  
        System.out.println("Game is running");  
        break;  
}
```


- Problem kod predstavljenog primera je što je i dalje radimo sa celobrojnim vrednostima

Implementacija enumeracije

- Seriju konstanti kojima je predstavljen status igre u prethodnom primeru, pomoću enumeracije možemo definisati na sledeći način:

```
public enum GameState {  
    Running, Paused, Stopped;  
}
```

- I kasnije koristiti veoma slično konstantama iz prethodnog primera



```
GameState state = GameState.Paused;  
switch(state) {  
    case Stopped:  
        System.out.println("Game is stopped");  
        break;  
    case Paused:  
        System.out.println("Game is paused");  
        break;  
    case Running:  
        System.out.println("Game is running");  
        break;  
}
```

- Vidimo da se u enumeraciji ne pominje broj, već samo ključna reč (polje)

Dobavljanje enumeracije na osnovu stringa

- U primeru smo videli da je enumeracija dobar način za predstavljanje nekog stanja. Često, ovo stanje ćemo hteti da učinimo perzistentnim (da ga sačuvamo u fajlu ili bazi podataka), i tada moramo sačuvati string ili broj
- Ako enumeraciju čuvamo u obliku stringa, možemo je kasnije rehabilitovati metodom `valueOf`, kao u sledećem primeru:

```
String persisted_state = "Running";  
GameState recovered = GameState.valueOf(persisted_state);  
System.out.println("Game state is " + recovered);
```

Podatak koji je stigao iz sistema koji ne zna za enumeraciju

Konverzija ulaznog podatka (stringa) u enumeracioni tip

Regularno korišćenje enumeracije

Dobavljanje liste enumeracija

- Enumeracija ima ugrađen sistem za iteraciju kroz vrednosti
- Ovo nam omogućava da vidimo sve vrednosti koje su nam na raspolaganju unutar enumeracije (ovo bi mnogo teže dobavili kroz konstante klase)
- Za dobavljanje niza svih vrednosti jedne enumeracije koristimo metod **values()**

```
System.out.println("Available game states");  
for (GameState st : GameState.values()) {  
    System.out.println("State: " + st);  
}
```

- Ukoliko hoćemo da tretiramo enumeracije kroz kolekcije. Na primer da smestimo samo odabrane vrednosti u neku kolekciju, ili nam je potreban samo opseg enumeracija (što će se verovatno jako retko dešavati), možemo koristiti posebnu implementaciju Set-a, klasu EnumSet

```
EnumSet states_range = EnumSet.range(GameState.Paused, GameState.Stopped);  
for (Object st : states_range) {  
    System.out.println("State: " + (GameState) st);  
}
```

Proširivanje enumeracionih objekata

- Do sada smo enumeraciju tretirali kao skladište konstanti, ali svaka vrednost enumeracije zapravo može biti daleko bogatija od obične konstante – može biti kompletan objekat
- Definiciju po kome će biti kreirani enumeracioni objekti kreiramo struktuiranjem same enumeracije

```
public enum FullGameState {  
    Running, Paused, Stopped;  
    public String state_description = "This is some state";  
}
```

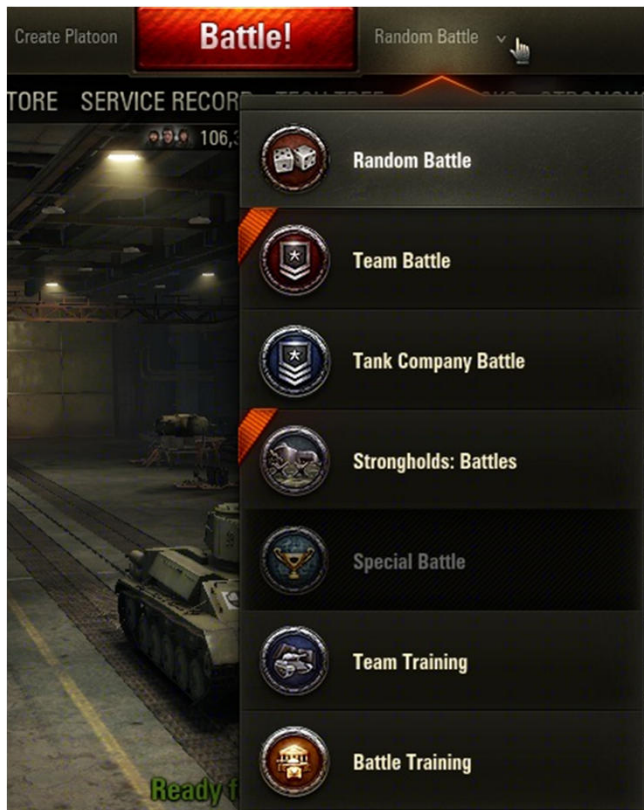
```
FullGameState fg_state = FullGameState.Running;  
System.out.println(fg_state.state_description);
```


Parametrizacija enumeracionih objekata

- Osim što svaki član enumeracije može nositi dodatne vrednosti (što smo videli na prethodnom slajdu), ove vrednosti mogu biti karakteristične za svakog člana
- Da bi povezali određene vrednosti sa određenim članovima, koristimo konstruktorsku parametrizaciju enumeracije

```
public enum FullGameState {  
    Running(1, "Game is running"), Paused(2, "Game is paused"), Stopped(3, "Game is stopped");  
    public int state_id;  
    public String state_description;  
    FullGameState(int id, String desc){  
        state_id = id;  
        state_description = desc;  
    }  
}
```

Vežba 4

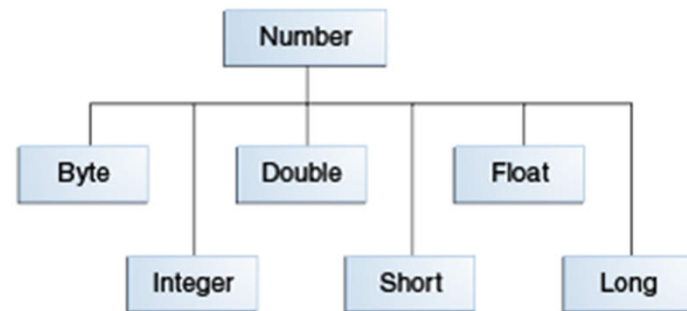


- Pokušajte da, na što bolji način pomoću enumeracije realizujete listu tipova borbe sa leve strane

Wrapperi prostih tipova

<https://docs.oracle.com/javase/tutorial/java/data/index.html>
(jcex112014 SimpleTypeWrappers)

- Java poseduje takozvane wrapper klase za sve primitivne tipove
- Ove klase nazivaju se kao i tipovi koje predstavljaju, ali sa velikim slovom na početku (Byte, Integer, Double, Short...)
- Klase su nemutabilne i mogu se koristiti na isti način kao i tipovi koje reprezentuju, ali takođe izlažu i dodatne funkcionalnosti (na primer, eksplicitnu konverziju i slično)
- Veći deo wrapper-a prostih tipova u Javi ima kao osnovu apstraktnu klasu **Number**



Inicijalizacija wrapper objekata

- Za razliku od ostalih objekata, wrapper objekat možemo kreirati običnom dodelom (bez eksplicitnog instanciranja)
- Takođe, tokom postojanja, objekat će se ponašati kao standardan prosti tip:

```
Integer my_integer = 10;  
my_integer += 25;  
System.out.println(my_integer);
```

- Objekat možemo takođe inicijalizovati pomoću konstruktora, pri čemu možemo direktno uneti inicijalnu vrednost
- Kod nekih konstruktora, moguće je čak uneti i string koji će biti implicitno konvertovan u odgovarajući tip (treba naravno biti jako pažljiv u ovakvim slučajevima)

```
Integer my_integer_obj = new Integer(10);  
System.out.println(my_integer);
```

```
Integer my_integer_from_string = new Integer("10");  
System.out.println(my_integer_from_string);
```

```
Integer my_integer_from_bad_string = new Integer("hello");  
System.out.println(my_integer_from_bad_string);
```

Korišćenje wrapper objekata

- Među dodatnim opcijama wrappera je i mogućnost eksplicitne konverzije tipa pomoću odgovarajuće metode
- Wrapper tipovi imaju i poseban metod za binarno poređenje (compareTo), mada je moguće porediti ih i pomoću standardnog operatora ==

```
int parsed_int = Integer.parseInt("150");  
System.out.println(parsed_int);
```

```
System.out.println(Integer.SIZE);  
System.out.println(Integer.MAX_VALUE);  
System.out.println(Integer.BYTES);
```

```
float x = my_integer.floatValue();  
double y = my_integer.doubleValue();  
byte z = my_integer.byteValue();  
int i = 35;  
System.out.println(my_integer.compareTo(i));  
System.out.println(my_integer==i);
```

Svaki wrapper prostog tipa ima ugrađen metod za eksplicitnu konverziju (u sopstveni tip)

Takođe, wrapperi imaju statička polja koja nose informacije o tipu

Wrapperi prostih tipova

LINKgroup

Klasa Math

<https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html>

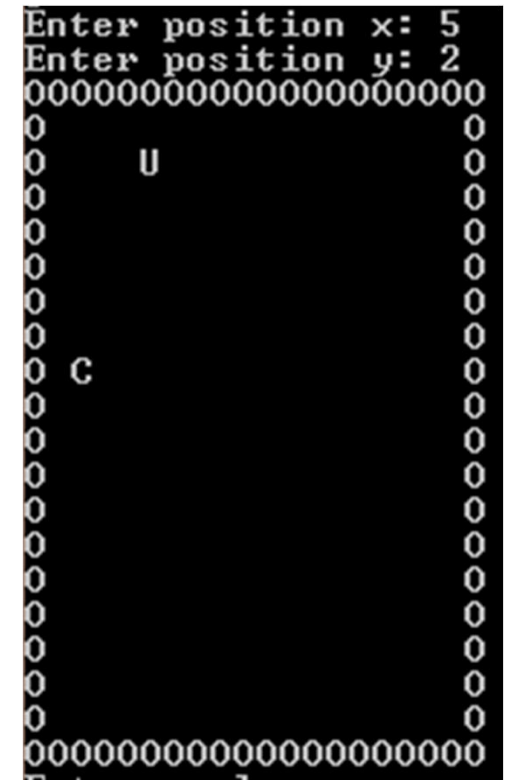
- Operatori predstavljeni do sada ne omogućavaju automatski kompleksnije matematičke operacije (stepenovanje, korenovanje, apsolutnu vrednost, zaokruživanje, dobavljanje sinusa i kosinusa)
- Ova klasa takođe sadrži statička polja sa vredostima PI i Eulerov broj

```
System.out.println(Math.ceil(0.5));  
System.out.println(Math.floor(0.5));  
System.out.println(Math.round(0.5));  
System.out.println(Math.sin(0.5));  
System.out.println(Math.cos(0.5));  
System.out.println(Math.pow(2, 3));
```

```
System.out.println(Math.PI);  
System.out.println(Math.E);
```

Vežba 4 (jcex112014 Battlefield)

- Potrebno je kreirati jednostavnu igru u kojoj igrač bira poziciju na zamišljenoj tabli veličine 10x10 tačaka
- Nakon igračevog odabira, poziciju bira računar
- Nakon što su pozicije odabrane, prikazuje se slika sa pozicijama igrača (samo konzolna skica), i to tako što se okvir predstavlja proizvoljnim karakterom, igrač karakterom U, a kompjuter karakterom C, kao na slici:



Nakon unosa ugla, prikazuje se putanja projektila (karakterom *) kao na slici desno

[illegible]

Ako se putanja ne pređe preko tačke na kojoj se nalazi slovo C, smatra se da je igrač promašio, i čeka se da računar izvrši „napad“

Računar zatim po slučajnom izboru bira ugao i ako pogodi igrača (poziciju slova U), igra se smatra završenom. U suprotnom, ponavlja se kompletna procedura biranja ugla, sve dok igrač ili računar ne pogode jedan drugog

Ukoliko dođe do pogotka, igra se završava i ispisuje se rezultat na izlazu, a zatim se ponavlja kompletna procedura (od odabira pozicija), kao na slici desno

[illegible]

```

You HIT computer
*****
Total shots      6
*****
Another game (y/n)?

```


Vežba 4 (pomoć)

- Da bi od ugla i brzine dobili x i y poziciju možemo da upotrebimo sledeću formulu:
 $x = \text{brzina} * \cos(\text{ugao});$
 $y = \text{brzina} * \sin(\text{ugao});$